Spring Boot and Spring Security. The service will provide real-time or near real-time stock quotes to client applications.

Mandatory Technologies:

- **Spring Boot:** Utilize Spring Boot for rapid application development and simplified configuration.

Requirements:

- **Data Source:** Connect to an external stock market data source (e.g., IEX Cloud, Alpha Vantage, Polygon.io) using appropriate libraries (e.g., RestTemplate, WebClient). You'll need to research and choose an API that provides stock quote data and create an account to obtain an API key.
- **Data Model:** Define a data model to represent a stock quote, including attributes like symbol, price, change, percentage change, timestamp, etc.
- **API Endpoints:** Develop RESTful API endpoints for the following functionalities:
    - **Get Quote by Symbol:** Allows retrieving a single stock quote based on the provided stock symbol.
    - **Get Batch Quotes by Symbols:** Allows retrieving quotes for multiple stocks at once by specifying a list of symbols.

Additional Considerations:

- **Error Handling:** Implement proper error handling mechanisms to gracefully handle situations like invalid symbols, API errors, and network issues.
- **Caching (Optional):** Consider implementing a caching mechanism using a library like Caffeine to optimize performance by storing frequently accessed quotes in memory.
- **Scalability:** Design your backend with scalability in mind, allowing it to handle a growing number of client requests.
- **Documentation:** Provide API documentation for your service, including endpoint descriptions, request parameters, response formats, and error codes.

Testing:

- Implement unit tests for your data model, API endpoints, and Spring Security configuration.
- Consider writing integration tests to simulate communication with the external data source.

Deliverables:

- Source code for the Spring Boot backend service.
- API documentation in a readable format (e.g., OpenAPI specification).
- Instructions on how to run the service. (Dockerfile or similar configuration)

Grading Criteria:

- Functionality (completeness of required features)
- Correct implementation of Spring Boot and Spring Security
- Code quality (clean code, modular design, appropriate use of Spring features)
- Documentation clarity and completeness
- Unit test coverage
- Consideration for scalability and performance
- Overall presentation and maintainability of the code

Tips:

- Explore Spring Boot starter dependencies for easier integration with the chosen data source and other libraries.
- Leverage Spring Security annotations and configuration options for securing API endpoints.
- Focus on writing clean, reusable, and well-documented code, especially for the security aspects.

Make sure you provide enough tests to validate the code