# Testing on “Advancing Market Integrity: A Proximal Policy Optimization Approach to Detect Spoofing and Layering in Algorithmic Trading”

Testing is a crucial phase in software development, applied at various stages from the early development phase to post-deployment. Its primary purpose is to ensure the software operates correctly, meets the specified requirements, and is free of defects. Effective testing improves software quality, enhances performance, and increases reliability by identifying and addressing bugs and issues early in the development process. By systematically validating and verifying the software, testing helps to prevent costly errors, maintain user satisfaction, and ensure compliance with standards and regulations. Overall, testing is essential for delivering robust and dependable software solutions.

The web application developed for my Bachelor's Thesis integrates advanced features to detect spoofing in algorithmic trading on the LUNA platform using Level 3 Limit Order Book (LOB) data. Built using Django REST Framework for the backend and React with TypeScript and Sass for the frontend, the application employs a Proximal Policy Optimization (PPO) model to identify spoofing activities. Key features include real-time data processing, advanced machine learning algorithms for anomaly detection, a user-friendly interface for visualizing trading patterns, and secure API endpoints for data interaction. This comprehensive integration aims to enhance the security and integrity of algorithmic trading by providing reliable and timely spoofing detection.

Unit testing focuses on verifying the smallest parts of the application, such as functions and methods, to ensure they work as intended. For the PPO model used in detecting spoofing, unit tests could be written to validate the correctness of the model's prediction function. This involves feeding the function with sample LOB data and verifying that the output matches expected spoofing detection results. Additionally, unit tests can ensure that the preprocessing steps for the LOB data, such as normalisation and feature extraction, perform correctly and handle edge cases gracefully.

Integration testing aims to test the interaction between different modules of the application to ensure they work together seamlessly. Key modules for integration testing in this application include the data ingestion module, the machine learning model, and the API endpoints. An effective integration strategy involves testing the data flow from the real-time LOB data ingestion to the PPO model's spoofing detection and finally to the frontend visualisation. This can be achieved by creating end-to-end test scenarios where sample data is processed through the entire pipeline, verifying that each module correctly passes data to the next and that the final output is accurate and timely.

System testing involves evaluating the entire application as a whole to ensure it meets the specified requirements and functions correctly in a realistic environment. Scenarios to be tested include the application's ability to handle large volumes of LOB data in real-time, the accuracy and performance of the PPO model in detecting spoofing activities, the responsiveness and usability of the frontend interface, and the security of data interactions through the API endpoints. Stress testing can also be conducted to determine the application's stability under peak loads, ensuring it can maintain performance and reliability during high-frequency trading periods.