Functional Programming tutor - Project Feedback Summary

Milestone 1.docx
Document

Please give feedback on my project attached.

Your project on "Cross-traded Crypto Currencies Identification" is well-structured, reflecting a good understanding of Domain-Driven Design (DDD) principles and workflows. Here are a few areas of strength as well as some suggestions for improvement:

Strengths:

1. **Clear Workflows**: Your workflows are clearly defined, with distinct input, output, and actions for each subdomain. This clarity helps in understanding how each part of the system functions, which is crucial for maintainability and debugging.
2. **Command and Event Separation**: You have effectively separated commands and events in your workflows, which aligns well with DDD practices. Commands (e.g., `StartPlacingOrder`) and events (e.g., `TradeCompleted`) are well-defined, showing a good understanding of how business processes are triggered and how results are handled.
3. **Usage of External Systems**: The way you've incorporated external systems (e.g., ExchangeService, Database, Polygon.io) is practical and shows consideration for real-world application. This approach fits well within the DDD framework where external systems play a role in bounded contexts.
4. **Detailed Data Structures**: The types and events (e.g., `CurrencyPairWithPriceQuote`, `OrderDetails`, `FulfillmentStatus`) are well-defined with appropriate details that map closely to the business domain. This provides a strong foundation for communication between components.

Suggestions for Improvement:

1. **Workflow Refinement**: While your workflows are defined well, a couple of them could benefit from further refinement. For instance, in the "Real-time Market Data Feed Check," it might help to expand the conditions for checking price spreads. Instead of just stopping when the minimal price spread value is met, consider additional branching to

handle different market conditions more robustly (e.g., handling unexpected interruptions in the data feed).

2. **Subdomain Classification**: You have correctly classified your subdomains as "Core" and "Supporting." However, you might want to reconsider the role of the "Real-Time Market Data Feed Check" and "Trading Strategy" workflows. Given their critical role in making trading decisions, they could be classified as core instead of supportive. This would help prioritize their implementation and refinement, which is vital for business success in a trading platform.
3. **Event Naming Consistency**: In some parts of the document, the event names are consistent and follow the past-tense convention, as per the DDD guidelines (e.g., `CrosstradedCryptocurrenciesIdentified`). However, some could be further refined for consistency and clarity. For example, `StartPlacingOrder` could be renamed to reflect an event in past tense like `OrderPlaced`. This would align better with event-driven architectures, where events are an expression of something that has already happened.
4. **Bounded Contexts**: It's important to ensure that all workflows are properly scoped within bounded contexts, which can be improved. For example, it would be helpful to explicitly draw boundaries between different contexts such as "Crypto-currency pair selection" and "Trading Strategy." Explicit boundaries will clarify where shared concerns (like notifications) end and domain-specific concerns begin.
5. **Error Handling**: Consider adding more details about how errors are handled across your workflows, especially domain errors. For example, in the "Order Fulfillment" workflow, specifying how the system handles partially fulfilled orders (e.g., retrying or notifying the user) would add robustness to your design. Adding this information would ensure that workflows can gracefully handle issues as they arise, rather than simply terminating.
6. **Side Effects and Functional Design**: Since you're applying DDD in the context of functional programming, isolating side effects, as discussed in class, is key. For instance, your interaction with external systems like the database should clearly separate the side-effect-free business logic from the impure operations like persisting data. You might want to reconsider workflows such as "Sell Order" and "Buy Order" to emphasize purity in the core business logic and isolate persistence as a final step.

Conclusion:

Overall, your project is well-executed, with clear alignment to DDD principles. Focusing on refining event consistency, bounded contexts, and error handling will enhance the clarity and

robustness of your design. Keep leveraging functional programming techniques to isolate side effects and ensure that your workflows remain pure at the core. Great work so far!