

ChatGPT

Your team project documentation appears well-organized, and the diagram captures different contexts involved in the system. I'll provide feedback and suggest improvements based on the key areas outlined in the project requirements.

1. Bounded Contexts and Domain Type Classification

- **Clarity:** The diagram clearly defines multiple bounded contexts like Cross-Traded Currency, Arbitrage Opportunity, Real-Time Feed Data, Transaction, etc. Make sure you explicitly indicate whether each context is **core**, **supporting**, or **generic** (e.g., core for Arbitrage Opportunity, supporting for Real-Time Feed Data).
- **Improvement:** Include the classification of domain types next to each context in the diagram or the documentation, to clearly indicate whether a context is core, supporting, or generic.

2. Events/Commands and Data Structures

- **Clarity:** Events and commands (yellow and blue boxes) are mapped well within the bounded contexts. Make sure that the data structures and the exchange mechanisms are thoroughly documented.
- **Improvement:** Ensure that each workflow step has clearly defined input and output, including data types or structures involved, especially where you're crossing between bounded contexts. For example, the interaction between the Real-Time Feed Data context and the Arbitrage Opportunity context should be annotated with a clear description of what data is passed (e.g., currency rates, trading signals).

3. Workflows

- **Completeness:** The workflows are generally coherent and align well with Domain-Driven Design. They clearly depict processing pipelines and inputs/outputs at various stages.
- **Improvement:** Ensure that each workflow is regrouped based on the bounded context they belong to. The diagrams seem clear, but regrouping them in the documentation will improve readability.
- **Suggestion:** Add steps or annotations for events that handle failures or domain errors to cover edge cases.

4. Domain Errors

- **Improvement:** Incorporate more detail on handling domain errors. For example, errors in the User Settings context like “Invalid Threshold” should have a clear output path or handling mechanism for fallback behavior.

5. External Systems and Interactions

- **Clarity:** External systems (e.g., user interactions or the database) are depicted but make sure they are annotated properly in the workflow.
- **Improvement:** Use @annotations (as mentioned in the project guidelines) to represent interactions with external systems, such as how signals are sent from the User Signal Context to an external system.

6. Domain Services

- **Optional:** If any domain services are employed (e.g., for validating trading signals), mention them explicitly with pseudo-code or a clear workflow step. This will enhance the completeness of the documentation.

7. Pseudo-Code with Ubiquitous Language

- **Improvement:** For your final submission, ensure that your pseudo-code notation uses the Ubiquitous Language of your domain (e.g., terms like execute trade, identify arbitrage). Avoid F# syntax and aim for clear, concise statements reflecting the real domain.
-

Suggested Corrections

1. **Bounded Context Classification:** Add specific classifications (core/supporting/generic) next to each context.
2. **Workflows:** Regroup workflows under each bounded context and provide detailed annotations of events and commands.
3. **External Systems:** Use @annotations to clarify interactions with external systems.
4. **Domain Errors:** Add more detail on how errors are handled within workflows.
5. **Clarity and Naming:** Ensure all names are consistent with the Ubiquitous Language and clearly defined to avoid ambiguity.

Make sure to take advantage of the GenAI model to refine the workflows, and note down any changes as required for the reflection module.

4o

You said:

針對下面的 requirement 紿我修正和改進 "Please give feedback on my project attached base on the following requirement. What is this application about? We call it ArbitrageGainer and the goal is to seize “arbitrage” opportunities which present themselves when there exists a price difference in cryptocurrencies on various cryptocurrency exchanges e.g Kraken and Bitstamp. Whenever a given cryptocurrency is available at a lower price on a certain exchange and at a higher price on the other, the application initiates an order to purchase on the low-priced exchange and an order to sell on the high-priced exchange. Like in real-world trading, both orders need to be matched with a seller/buyer on the respective stock exchanges. As markets move swiftly this is not guaranteed to happen hence every arbitrage transaction has inherent risk to it. What are the application’ s high-level features? A system user should be able to do the following: - Provide several user-defined trading strategy parameters: number of crypto currencies to track, minimal price spread value, minimal transaction profit, maximal transaction value, maximal trading value (across a set of all transactions), initial investment amount (teams of 4 only). The user needs to be able to provide his e-mail for notification purposes and reset the maximal trading value. - On demand, retrieve a list of crypto currencies traded at selected cryptocurrency exchanges (Bitfinex, Bitstamp, Kraken) and identify currency pairs that are traded across multiple exchanges (algorithm provided below). - On demand for all the cross-traded crypto currencies and a given time period, determine how many historical arbitrage opportunities occurred (algorithm provided below). - Launch real-time trading activity. This requires subscription to real-time market data feed (Polygon.io) for crypto currencies that exhibited price spread in the past, retrieving the data continuously and applying user’ s trading strategy automatically until stop conditions are met (algorithm provided below). - Stop real-time trading activity - a user can choose to manually stop the trading activity at any time. Teams of 4 have to implement an auxiliary module calculating Profits & Losses as well as Annualized return metric (see last page). Are there any assumptions concerning the system? - System works for 1 user only (it’ s like a personal trading app) - System deals with 1 currency only: \$ (dollars) Cross-traded crypto currencies identification algorithm 1. Retrieve a list of crypto currencies traded at every exchange (Bitfinex, Bitstamp, Kraken) separately. 2. Compare the sets to identify crypto currency pairs that are traded at at least two exchanges. 3. Store the information on cross-traded cryptocurrencies in the database in the following format: currency1Symbol-

currency2symbol (e.g. CHZ-USD). • The objective is to identify all cross-traded currency pairs. Very often they trade against USD only but if crypto-crypto pair is sold on more than 1 exchange then it should also be accounted for. • You can treat currency pairs as a set of 3 letter tickers and ignore anything else - e.g. XYZABC would correspond to currency1 = XYZ and currency 2 = ABC. In the case of pairs where one currency has >3 letters (e.g. AUDIOPUSD), you should ignore this pair altogether. Historical arbitrage opportunities calculation algorithm 1. Load historical values file (text file with quotes). 2. For all data points (regardless of their origin – underlying currency exchange): 1. Regroup quotes into buckets of 5 milliseconds (no need to adjust the groupings relative to the timestamp of the first quote). 2. For each crypto currency pair where there is data from more than 1 exchange: 1. Select a quote with highest bid price for every exchange. 2. Compare bid and ask prices between all exchanges and identify arbitrage opportunity (identified as a price difference of more than 0,01\$). 3. Comparison should be carried out in both direction (e.g. ask in exchange 1 to bid in exchange 2 and ask in exchange 2 to bid in exchange 1). 4. Should there be multiple pairs of quotes from the same exchanges that meet the criteria, only 1 opportunity needs to be accounted for. 3. Persist the arbitrage opportunities in the following format: currency1Symbol-currency2symbol (e.g. CHZ-USD); numberOfOpportunitiesIdentified (e.g. 20) Real-time trading algorithm 1. Subscribe to real-time data feed (Polygon.io) for a set of cryptocurrency pairs (defined by “number of crypto currencies to track” trading parameter) that exhibited most arbitrage opportunities in the past (outcome of historical arbitrage opportunities calculation) and retrieve the data continuously for selected exchanges. • The cryptocurrency pairs need to be cross-checked with the list of cross-traded crypto currencies. 2. Process continuously real-time feed for a given currency pair to identify arbitrage opportunities: 1. Upon receiving new price quote for a subscribed currency pair, check if a price spread exists (defined by “minimal price spread value” trading parameter). 2. If so, apply a trading strategy which takes into consideration the following user-defined parameters: o minimal profit, o maximal total transaction value = buy + sell orders value, o maximal trading value. 3. If the arbitrage opportunity is deemed worthwhile by the trading strategy algorithm, emit buy and sell market orders to the concerned cryptocurrencies exchanges. • The quantity of cryptocurrency units purchased in a buy order should match the quantity of the same cryptocurrency units available for sale at another exchange. • The buy order quantity should not exceed the ask quantity but can be smaller. • The system should act only on arbitrage opportunity with highest spread (if arbitrage is possible across multiple exchanges) and emit 1 buy and 1 sell order. • If the available quantity of cryptocurrency at a specific price on an exchange is not fully utilized in a single order, the remaining quantity should be retained for potential use in subsequent orders. • If the arbitrage opportunity exceeds maximal total transaction value or maximal

trading value, the order quantity should be reduced to meet the applicable limits (so that the opportunity is acted on, while respecting the trading strategy). 4. Persist the orders emitted in the database. 3. Retrieve the order status updates:

- For both order legs (buy and sell) fully fulfilled, store transactions history in a database.
- For partially fulfilled orders, emit one more order with the remaining amount (desired amount – booked amount) and the original price; store the realized transactions and the newly emitted order in the database.
- For orders that had only one side filled, notify the user via e-mail and persist the transaction history in the database.

ChatGPT said:

ChatGPT

Memory updated

Here's a detailed analysis and feedback based on the provided requirements for your ArbitrageGainer project:

1. High-Level Features

- **User Parameters:**
 - You've outlined several key parameters (e.g., number of cryptocurrencies, minimal price spread, etc.), but ensure that the system allows users to easily input and update these parameters through a user-friendly interface.
 - **Improvement:** Document in the system design where and how these parameters will be stored and retrieved, especially for the **maximal trading value** reset feature. Ensure the storage format is efficient and accommodates real-time updates.
- **Cross-Traded Crypto Retrieval:**
 - The requirement specifies retrieving a list of cryptocurrencies traded across multiple exchanges. It's important to specify the exact flow for how the comparison between exchanges (e.g., Kraken and Bitstamp) will be done.
 - **Improvement:** Clarify in your documentation whether the algorithm will run continuously in the background or on demand (user-triggered). This will affect both the design and workflow.
- **Historical Arbitrage Opportunities:**
 - This feature focuses on identifying arbitrage opportunities based on historical data.

- **Improvement:** Ensure that the workflow for loading historical data from the file is well-defined. Mention if there are any limitations (e.g., how large the data file can be, time windows for data analysis).
- **Real-Time Trading Activity:**
 - This is the core of your application. Your current documentation covers it well.
 - **Improvement:** It would help to map out edge cases, such as handling issues when a trade isn't fully fulfilled (i.e., orders are only partially matched). Be specific in your pseudo-code about how the system handles these scenarios (e.g., partial order fulfillment, retries, or notifications).

2. Assumptions

- You've clearly mentioned that the system is for a single user and deals in dollars. No additional adjustments are necessary here, but make sure that this assumption is reflected in all relevant areas of your workflow.

3. Cross-Traded Crypto Identification Algorithm

- The algorithm itself seems straightforward. However, ensure that you address the following:
 - How the **storage of cross-traded pairs** will be optimized (e.g., indexes or a database table for quick lookups).
 - **Improvement:** Document how often this algorithm should run and under what conditions it needs to be updated (e.g., adding/removing exchanges).

4. Historical Arbitrage Calculation Algorithm

- This part of the system seems well-documented. A few improvements:
 - Ensure that quotes from different exchanges are **aligned** properly for meaningful comparisons.
 - **Improvement:** Clarify how the system deals with **latency** in historical data and how the regrouping into 5ms buckets will be managed in terms of timestamp precision.
 - Persisting the opportunities is clear, but mention where and how (e.g., database schema, key-value store) this data will be stored.

5. Real-Time Trading Algorithm

- **Order Execution Flow:**

- The algorithm for issuing buy/sell orders based on price spread is well-covered.
 - **Improvement:** Ensure that you describe how the **order execution priority** is determined. For example, if multiple arbitrage opportunities are found, how does the system decide which to execute first? Is it purely based on the highest spread?
- **Partial and Full Order Handling:**
 - Handling fully fulfilled orders seems clear, but the retry mechanism for partially fulfilled orders needs more elaboration.
 - **Improvement:** In your pseudo-code, explicitly document how many retries are allowed for partially fulfilled orders and the conditions under which the system abandons such orders.
- **Maximal Transaction and Trading Value Limits:**
 - Ensure the **logic for reducing order size** is clear when these limits are breached. This involves checking both the **maximal transaction value** and **maximal trading value** in real-time.
 - **Improvement:** Clarify how these limits are enforced without compromising trading speed.

6. External Systems (Polygon.io & Emails)

- **Real-Time Feed Integration:**
 - You mentioned using Polygon.io for real-time data. Ensure that you document how the system **subscribes and retrieves** real-time quotes. What is the **error-handling** strategy if the data feed is interrupted?
 - **Improvement:** Consider edge cases like network failures or feed delays. Clearly outline recovery strategies, such as fallback mechanisms or switching to historical data temporarily.
- **Email Notifications:**
 - It's important to document how and when email notifications are sent (e.g., partially fulfilled orders, maximal trading value breached).
 - **Improvement:** Ensure that email notifications are prioritized during trading activities so users remain informed in real-time.

General Feedback:

- **Completeness:** Your current design captures most high-level features, but ensure all algorithms (especially those dealing with historical data and real-time order fulfillment) are fully fleshed out in the documentation.

- **Coherence and Correctness:** The design is coherent, but add more detail around how data is stored and how the system prioritizes arbitrage opportunities when multiple trades are possible.
- **Clarity:** Add more clarity to domain-specific language used in your pseudo-code (ensure it's using Ubiquitous Language as required by Domain-Driven Design principles).

Once you make these adjustments, your documentation will better align with the requirements, and it will be easier for the evaluator to assess the completeness and correctness of your solution.

I wonder What does "Data Structures" refer to in Domain-Driven Design in the Mile stone one?

ChatGPT said:

ChatGPT

Here's a refined version of your question:

Question Title: What Does "Data Structures" Refer to in Domain-Driven Design for Milestone One?

Question Content:

In the context of **Milestone One** for my Domain-Driven Design (DDD) project, we are required to list **Data Structures**. I'm a bit unclear on what exactly is meant by "Data Structures" here. Does it refer to the format of the data used in events and commands, or does it involve a more detailed explanation of how data is structured and exchanged between bounded contexts and external systems?

Any clarification would be appreciated!

This phrasing will help you get specific guidance related to Milestone One in your DDD project.

You said:

I wonder What does "Data Structures" refer to in Domain-Driven Design in the Mile stone one? Does it mean we need to explicit each data type? ex. string, list?

ChatGPT said:

ChatGPT

Here's a refined version of your question:

Question Title: What Does "Data Structures" Refer to in Domain-Driven Design for Milestone One? Do We Need to Specify Data Types?

Question Content:

In **Milestone One** of my Domain-Driven Design (DDD) project, we are asked to list **Data Structures**. Does this mean we need to explicitly define each data type (e.g., string, list, etc.), or is it more about providing an overview of how data is structured and exchanged between bounded contexts? I want to make sure I'm including the right level of detail.

Thanks for any clarification!

This version directly asks whether you need to specify data types, ensuring clarity on what level of detail is expected.

40

You said:

B. Events, Commands, Data Structures, Exchange Mechanisms, and Domain Errors Across Bounded Contexts and External System. Events, Commands, External Systems and error

ChatGPT said:

ChatGPT

Your revised title "**Definition of Events, Commands, External Systems, and Errors**" is almost correct, but it omits **Data Structures** and **Exchange Mechanisms**, which were part of the original context. If you're intending to focus only on **Events, Commands, External Systems**, and **Errors**, and exclude the other elements, your title is fine.

However, if you still want to encompass the full scope, including data structures and exchange mechanisms, a better title might be:

"Definitions of Events, Commands, External Systems, and Domain Errors"

If you intend to shorten it while retaining the essential elements, this title would be appropriate.