Assignment 3

Event sourcing

> Time is nature's way of keeping
> everything from happening at once.
>
> *John Archibald Wheeler*

Design and implement a basic trading system using event sourcing. All system state — including orders, trades, and balances — must be reconstructed by replaying a sequence of immutable domain events. This approach mirrors real-world financial systems that prioritize auditability and historical accuracy.

To do

1. **Event Types**: Define 5 event types representing domain actions:
   - `OrderPlaced` represents a new buy or sell order submitted by a user. When replayed, it adds the order to the order book.
   - `OrderCancelled` indicates that a user has cancelled a pending order. When replayed, it marks the order as cancelled and removes it from the active order book.
   - `TradeExecuted` signals that a buy and a sell order have matched, and a trade has occurred.
   - `FundsDebited` records that a user's funds have been deducted (e.g. for buying shares). When replayed, it decreases the user's available balance.
   - `FundsCredited` records that a user's account has received funds (e.g. from selling shares). When replayed, it increases the user's available balance.
2. **Event Store**: Build an in-memory or file-backed event log that supports the following API:
   - `append(event)` , `get_all_events()`
3. **Command Handlers**: Accept commands (e.g., `place_order` , `withdraw_funds`) and emit appropriate events — commands do not directly mutate state.
4. **Aggregate Reconstruction**: Implement one or more domain models (e.g., `OrderBook` , `Account`) that rebuild their state by replaying events.
5. **Replay Support**: Reconstruct the full application state from scratch using only the event log.

Assignment 3

Further reading

1. Martin Fowler's Article on Event Sourcing
2. Intro to Event Sourcing (Microsoft)
3. Intro to CQRS (Microsoft)

Important: The deadline for this assignment is Week 11.