Document 2: Bonus Features Documentation

Cryptocurrency Matching Engine - Bonus Features

Advanced Order Types

The system supports multiple advanced order types that mimic real-world trading behavior. These additional order types enhance flexibility for clients and traders.

a) Immediate-Or-Cancel (IOC)

- IOC orders execute as much quantity as possible immediately.
- Any unfilled portion is automatically canceled.
- Useful for traders who want instant partial fills without resting on the book.

Example Flow:

- Client submits IOC SELL 0.5 BTC @ 46000.
- If only 0.3 BTC is available at that price, 0.3 is filled and 0.2 is canceled.

Code Logic:

```
if (order.type == OrderType::IOC) {
  match(order);
  // remaining() automatically canceled if unmatched
}
```

b) Fill-Or-Kill (FOK)

- FOK orders must either fill completely or be rejected entirely.
- Useful for block trades and liquidity-sensitive algorithms.

Example Flow:

- Client submits FOK SELL 1 BTC @ 46000.
- If full 1 BTC liquidity exists, order fills entirely.
- If not, entire order is rejected.

Code Logic:

```
if (order.type == OrderType::FOK) {
```

```
  bool fullyFillable = checkFullLiquidity(order);
  if (fullyFillable) match(order);
  else reject(order);
}
```

c) Market Orders

- Executes against the best available prices until quantity is fully filled or book is exhausted.
- May result in multiple price levels being consumed.

Code Logic:

```
if (order.type == OrderType::MARKET) {
  match(order, priceIgnore=true);
}
```

Persistence System

Design Goals:

- Complete audit trail.
- Crash recovery.
- Replayability of historical events.

Journal File:

- Events are written to journal.log.
- JSON-lines format (one event per line).
- Uses PersistenceManager class.

Events Logged:

- NEW (order accepted)
- RESTED (order entered book)
- PARTIAL_FILL
- FILLED
- CANCELED

Example Log Entry:

{"event": "PARTIAL_FILL", "orderId": "o101", "side": "SELL", "price": 46000, "quantity": 0.05}

Recovery Possibility:

- On startup, journal can be replayed to rebuild full order book state.
- This enables crash recovery and audit compliance.

Implementation:

- Mutex protected journal file writes.
- Immediate flush() after each write ensures durability.
- Non-blocking to matching performance.

Maker-Taker Fee Model

Fee Calculation Model:

- Each trade applies fees depending on whether the participant is a maker or taker.
- Fee rates are configurable.

Role	Fee Applied
Maker	0.1%
Taker	0.2%

Fee Calculation Code:

```
struct FeeModel {
 double makerRate = 0.001;
 double takerRate = 0.002;

 FeeResult computeFees(double price, double qty, bool isMaker) const {
  double gross = price * qty;
  return FeeResult {
   .makerFee = gross * makerRate,
   .takerFee = gross * takerRate
```

```
  };
 }
};
```

TradeReport Includes Fees:

```
{
 "price": 46000,
 "quantity": 0.1,
 "makerFee": 4.6,
 "takerFee": 9.2
}
```

Benefits:

- Accurately models real-world exchange revenue models.
- Maker fees incentivize liquidity provision.
- Taker fees discourage aggressive trading.

Summary Table of Bonus Features

Feature	Benefit
IOC	Allows partial instant fills
FOK	Enforces full fills or rejects
MARKET	Full consumption of liquidity
Persistence	Durable audit logging
Journal	Enables crash recovery
Fee Model	Revenue simulation with configurable fees
