# **Reflection and Rationale**

## **Design Choices**

### **a. Major Design Decisions**

#### **1. Class-Based Architecture with Express Integration**

I chose to implement the core logic in a Promotion Engine class separate from the Express server configuration and the route handler logic. This provides a clear separation of concerns:

* **Business Logic**: Contained within the class for easy testing and reuse
* **API Layer**: Express handles HTTP concerns (routing, middleware, error handling)
* **Testability**: The class can be unit tested independently of the web framework

**Alternative Considered**: A purely functional approach with separate modules. I rejected this because the rule engine maintains state (loaded rules, metrics) that naturally fits an object-oriented design.

#### **2. Priority-Based Rule Resolution**

Rules are sorted by priority at load time and evaluated in descending order, with the first match that is found winning. This approach:

* **Ensures Predictability**: Clear resolution of conflicting rules
* **Optimizes Performance**: Higher priority rules are checked first
* **Simplifies Configuration**: Business users can easily understand rule precedence

**Alternative Considered**: Evaluating all rules and choosing based on a scoring technique. I rejected this as it would be less predictable and harder to debug.

**5. Extensibility Hook Pattern**

I implemented three key extensibility points as separate methods:

* applyWeightedRandomness(): For probabilistic rule selection
* applyABTesting(): For player segmentation
* applyTimeWindows(): For temporal rule activation

This pattern allows future enhancements without modifying core evaluation logic.

### **b. Data Structures**

#### **1. Rule Array with Priority Sorting**

Rules are stored as a sorted array rather than a hash map because:

* **Evaluation Order Matters**: Priority-based selection requires ordered iteration
* **Memory Efficient**: No additional indexing overhead
* **Simple Implementation**: Straightforward to understand and debug

#### **2. Metrics Object**

I used a simple object to track metrics rather than a more complex system:

* **Low Overhead**: Minimal performance impact on rule evaluation
* **Sufficient Detail**: Captures key performance indicators
* **Easy Extension**: Additional metrics can be added easily

## **Trade-offs**

### **a. Memory Usage vs. Code Complexity**

**Decision**: Keep all rules in memory rather than implementing lazy loading.

**Trade-off**: Higher memory usage but simpler code and faster evaluation.

**Justification**: Rule sets are typically small (dozens to hundreds of rules), so memory impact is minimal while code simplicity aids maintainability.

### **b. Validation Strictness vs. Robustness**

**Decision**: Lenient validation that allows rule evaluation to continue with missing data.

**Trade-off**: More robust operation, but potentially masks data quality issues.

**Justification**: Gaming systems often have incomplete player data, so graceful degradation is more valuable than strict validation.

## **Areas of Uncertainty**

### **a. Rule Conflict Resolution Strategy**

I was initially uncertain about how to handle multiple matching rules. I considered several approaches:

1. **Return all matches**: More flexible but complicates client logic
2. **Score-based selection**: More sophisticated but less predictable
3. **Priority-based selection**: Simple and predictable

I chose priority-based selection because it's easy to understand and debug, which is crucial for business rule systems.

### **b. Error Handling Granularity**

I was unsure whether to fail fast on invalid player data or continue with partial data. I chose to continue evaluation with missing fields because:

* Gaming systems often have incomplete player profiles
* Partial matches are better than no promotion
* The fallback rule ensures some promotion is usually available

### **c. Hot Reload Safety**

I was uncertain about thread safety during rule reloading in a production environment. My current implementation replaces the entire rule array atomically, which is safe for Node.js's single-threaded model but might need refinement for multi-threaded environments.

## **AI Assistance**

### **Tools Used**

I used **Claude's Sonnet 4** to assist with this assignment in the following ways:

#### **a. Documentation and Examples :** *I prompted with a very detailed and customised request, which would result in documentation that was very curated with my needs for the Readme.md*

* Claude helped structure the README with appropriate examples
* Assisted in creating comprehensive cURL examples.
* Provided guidance on technical documentation best practices.

**b. Generating the rules.yaml sample file:** *I prompted with all the requirements mentioned in the assignment and my logic of using priority sorting based on conditions.*

* Claude helped be customise the rules.yaml file according to the given scopes in the assignment and my logic.
* It generated new properties in the rules to overall enhance the matching of the promotion with respect to the given properties.

**c.** **Writing some of the testcases:**  *I prompted with all the edge cases I could think of and general testcases, and requested to verify and generate more testcases that I couldn’t think of.*

* Claude helped me think out of the box to generate and run test cases to ensure that the endpoints don't crash.

All of the outputs from the AI were modified according to my needs before adding it to my assignment solution.

### **Areas Where I Relied on My Own Judgment**

1. **Business Logic**: The specific promotion rules and conditions came from my understanding of gaming monetization
2. **Performance Optimization**: The decision to pre-sort rules by priority was my optimization choice
3. **API Design**: The specific endpoint structure and response formats reflect my API design preferences
4. **Extensibility Points**: The choice of which extensibility hooks to implement was based on my understanding of common gaming platform needs

The AI assistance was invaluable for accelerating development and ensuring code quality, but the core architectural decisions and business logic design were driven by my analysis of the requirements and understanding of the gaming domain.