Project Report: High-Performance API for Access Validation

Project Summary

The project aims to build a robust, scalable, and efficient Access Validation API capable of handling 1 million requests per minute. This API validates user access for a gaming platform by evaluating requests against pre-defined rules. These rules are cached for optimal performance, and the backend leverages the Go programming language to ensure speed and concurrency.

Key features include:

1. Rule-based access validation using predefined criteria such as country, app version, platform, and app type.
2. High scalability through optimized Redis caching and concurrent request handling.
3. Comprehensive test coverage to ensure functionality and correctness.

The solution is designed with modularity, maintainability, and scalability at its core, making it ideal for high-traffic applications.

Challenges and Solutions

1. **Scalability Requirements**
    - **Challenge:** Meeting the target of 1 million requests per minute while ensuring consistent performance.
    - **Solution:**
        - Implemented Go's lightweight goroutines for concurrency.
        - Utilized Redis as a high-speed in-memory cache to store frequently accessed rules, reducing database queries.
        - Adopted a distributed architecture, allowing horizontal scaling by deploying multiple API instances behind a load balancer.
2. **Complex Rule Matching**
    - **Challenge:** Efficiently validating access requests against multiple complex rules.
    - **Solution:**
        - Cached rules in Redis to minimize latency.
        - Designed a validation logic that filters inactive rules and sequentially matches request attributes (e.g., version, country, platform) to applicable rules.
3. **Error Handling and Resilience**
    - **Challenge:** Handling invalid inputs, malformed requests, or system-level failures.
    - **Solution:**
        - Implemented comprehensive input validation.
        - Provided detailed error responses for invalid requests.
        - Added retry logic and timeouts to Redis operations to ensure robustness.
4. **Testing and Validation**

    - **Challenge:** Ensuring the system behaves correctly under diverse scenarios.
    - **Solution:**
        - Developed unit tests for all core functionalities, including rule validation and Redis caching.
        - Simulated edge cases such as missing rules and unsupported platforms.
        - Conducted performance testing to validate scalability goals.
5. **Configuration Management**
    - **Challenge:** Seamlessly managing configurations across development and production environments.
    - **Solution:**
        - Used environment variables for sensitive configurations such as Redis URLs.
        - Adopted a 12-factor app methodology for configuration management.

Conclusion

This project successfully meets its objectives by delivering a high-performance Access Validation API. It addresses critical challenges through innovative solutions in caching, concurrency, and error handling. With its scalable architecture and robust implementation, the API is well-suited for real-world, high-demand applications.