Software Test Plan (STP) - API Rate Limiter

Project: API Rate Limiter
Version: 1.1
Authors: Raunak Bagaria (PES1UG23CS475), R Jeevith (PES1UG23CS457), Rishav Ghosh (PES1UG23CS478), Roshit Sharma (PES1UG23CS489)
Date: 03-11-2025
Status: Approved

1. Introduction

Purpose:
This test plan defines testing objectives, scope, strategy, resources, schedule and responsibilities for API Rate Limiter v1.0 — the middleware service that enforces configurable rate limits on API endpoints to prevent abuse and protect backend services.

Scope:
Testing scope covers the Rate Limiter's functional modules: client identification (API key, IP), rate enforcement (multi-window and algorithmic enforcement), analytics & monitoring (headers, alerts, dashboard), configuration management (dynamic rule updates, validation, precedence), request forwarding behavior and security controls (MFA/RBAC, encryption, logging). Excluded: backend application logic of protected services, third-party monitoring tools, and underlying datastore implementation correctness (assumed vendor-tested).

References:

- Codevengers SRS — API Rate Limiter v1.0.
- Design Specifications v1.0 (project repo)
- Security & TLS standards (TLS 1.2+), encryption at rest (AES-256).

Definitions:

- **API** — Application Programming Interface.
- **Rate Limiting** — Controlled restriction of API request rates within configured time-windows.
- **Token Bucket / Leaky Bucket / Sliding Window** — supported limiting algorithms.
- **Client** — application or service consuming APIs via the limiter.
- **Admin** — operator managing limiter policies, dashboards and exceptions.

2. Test Items

- Client Identification module (API key, IP, allow/block lists)
- Rate Limit Enforcement module (multi-window, algorithms, per endpoint/IP/API key)
- Request Forwarding module (proxying with header/body preservation)
- Analytics & Monitoring module (alerts, headers, dashboard counters)
- Configuration Management module (dynamic updates, validation, precedence, blocking, error customization)
- Security Controls (MFA/RBAC, encryption, logging, anonymization)

3. Features to be Tested

Features mapped to SRS requirement IDs:

- **RL-F-001**: Identify clients using API keys
- **RL-F-002**: Identify clients using IP-based matching (CIDR ranges)
- **RL-F-003**: Support IP allowlists and blocklists
- **RL-F-004**: Enforce multi-window rate limits (second, minute, hour, day)
- **RL-F-005**: Support different rate-limiting algorithms (token bucket, sliding window, fixed window)
- **RL-F-006**: Allow rate limits per endpoint, IP, or API key
- **RL-F-007**: Forward approved requests while preserving headers and body
- **RL-F-008**: Trigger alerts when clients exceed 80% of their rate limits
- **RL-F-009**: Provide custom response headers with rate-limit information
- **RL-F-010**: Display real-time request counters on the admin dashboard
- **RL-F-011**: Support dynamic rule updates without service restart
- **RL-F-012**: Validate configuration syntax and semantics before applying changes
- **RL-F-013**: Enforce rule precedence with client-specific overrides
- **RL-F-014**: Allow manual blocking of sources by an administrator
- **RL-F-015**: Allow custom error messages for blocked/throttled requests
- **RL-NF-001**: Maintain latency overhead <= 10ms per request
- **RL-NF-002**: Provide 99.9% system availability
- **RL-NF-003**: Store and retrieve request count data per client and endpoint
- **RL-NF-004**: Maintain accuracy of rate limiting within ± 5% deviation
- **RL-NF-005**: Handle at least 1000 requests per minute under peak load
- **RL-SEC-001**: Require MFA and RBAC for administrative APIs
- **RL-SEC-002**: Log and alert on rate-limit bypass attempts
- **RL-SEC-003**: Encrypt sensitive data (API keys, tokens) at rest with AES-256
- **RL-SEC-004**: Validate request signatures to prevent tampering
- **RL-SEC-005**: Anonymize client IPs and identifiers beyond 30 days

4. Features Not to be Tested

- Backend application logic of protected APIs (assumed tested by backend owners).
- Underlying datastore implementation correctness (Redis/Memcached) — only integration and failure handling will be tested.
- Third-party monitoring/alerting platform internals (e.g., Prometheus, ELK stack) — only integration points validated.
- External API gateway features (routing, load balancing, authentication beyond rate limiting).
- Network infrastructure outside the controlled test environment.

5. Test Approach / Strategy

Levels:
- Unit tests (module-level)
- Integration tests (Client -> Rate Limiter -> Backend API)
- System tests (end-to-end rate limiting functionality)
- Acceptance tests (UAT)

Types:
- Functional testing (core features)
- Regression testing
- Performance testing (response time, load)
- Usability testing (UI clarity, accessibility)

Entry Criteria: Stable build delivered, test data available, test environment ready.
Exit Criteria: 100% of planned test cases executed, 0 critical defects open, all acceptance criteria satisfied.

5.1 Security Validation

- **API Key & Client Identification** – Validate secure handling, parsing, and anonymization of client identifiers.
- **TLS 1.2+ verification -** Ensure all communications (client ↔ limiter, admin ↔ limiter, limiter ↔ backend) are encrypted.
- **Configuration Security** – Validate encryption of sensitive configuration data (AES-256 at rest)
- Verify MFA + RBAC for admin APIs.
- Penetration testing of authentication flows and assess exposure of usage metrics and request logs.

6. Test Environment

Hardware: Cloud VM or local PC.

Software: API Rate Limiter app v1.0, mock backend API.

Tools: Postman , JMeter , Jira.

Test Data: Sample API keys, 10-20 test requests, basic config file.

7. Test Schedule

Milestones:
- Test case design: 05-Sep-2025
- Environment setup: 07-Sep-2025
- Test execution start: 08-Sep-2025
- Test execution end: 20-Sep-2025
- UAT: 22-Sep-2025 to 25-Sep-2025

8. Test Deliverables

- Test Plan (this document)
- Test Cases (manual & automated)
- Test Scripts
- Test Data
- Test Execution Logs
- Defect Reports
- Test Summary Report

9. Roles and Responsibilities

Role	Name	Responsibility
QA Lead	Rishav Ghosh	Prepare plan, coordinate execution
Test Engineer	Roshit Sharma	Design & execute test cases, log defects
Developer	Raunak Bagaria	Support defect fixes and triage
Product Owner	R Jeevith	Approve test results, sign-off readiness

10. Risks and Mitigation

Risk	Mitigation
Delay in stable build delivery	Request early smoke builds from dev team
Test environment downtime	Maintain backup environment on cloud VM
Performance bottlenecks under load (latency >10ms, throughput <1000 requests/min)	Run early load/performance tests with synthetic traffic

11. Assumptions & Dependencies

Assumptions: Mock backend API works; test data (API keys, requests).

Dependencies: API Rate Limiter app build is stable; basic network access available.

12. Suspension & Resumption Criteria

Suspend Criteria:

If the test environment is unavailable for more than four consecutive hours, testing must be suspended to prevent invalid or incomplete results
If the delivered build quality is too unstable, blocking more than 30% of planned test cases from execution, testing is suspended until the issues are addressed..

Resume testing if:
- Testing will resume once critical defects have been fixed and verified.
- The test environment must be restored to a stable and functional state, ensuring smooth execution.

13. Test Case Management & Traceability (not updated yet)

RTM ensures mapping of SRS requirements to test cases.
Example:
- ATM-F-001 (PIN validation) → TC-Auth-01, TC-Auth-02
- ATM-F-010 (Withdrawal) → TC-WD-01, TC-WD-02
- ATM-NF-001 (Response time) → TC-Perf-01

14. Test Metrics & Reporting

Metrics collected:
- % test cases executed
- % passed/failed
- Defect density

- Defect aging
- Requirement coverage

Reports:
- Daily execution status
- Final Test Summary Report

15. Approvals

Role	Name	Signature / Date
QA Lead	Rishav Ghosh	Rishav (03-11-2025)
Dev Lead	Raunak Bagaria	Raunak (03-11-2025)
Product Owner	R Jeevith	Jeevith (03-11-2025)

16. Test Cases

This section lists the test cases for the API Rate Limiter.

Test Case ID	Test Scenario / Description	Preconditions	Test Steps / Input Data	Expected Result	Postconditions / Remarks
TC-RL-01	Enforce per-second rate limit for free tier	Rate limiter initialized with default limits; client identified as free tier	1. Make 1 request as free tier client 2. Immediately make second request within same second	First request allowed (200 OK); Second request blocked (429 Too Many Requests)	Free tier per-second limit is 1 request
TC-RL-02	Enforce per-minute rate limit	Rate limiter initialized; client tier = free (limit: 10/minute)	1. Make 10 requests within 60 seconds 2. Make 11th request within same minute	First 10 requests allowed; 11th request blocked with 429 status	Minute window correctly enforces limit

TC-RL-03	Enforce per-hour rate limit	Rate limiter initialized; client tier = free (limit: 100/hour)	1. Make 100 requests within 60 minutes 2. Make 101st request within same hour	100 requests allowed; 101st request blocked	Hour window tracking works correctly
TC-RL-04	Enforce per-day rate limit	Rate limiter initialized; client tier = free (limit: 1000/day)	1. Make 1000 requests within 24 hours 2. Make 1001st request within same day	1000 requests allowed; 1001st blocked with retryAfter time	Day window enforced correctly
TC-RL-05	Different tier limits - premium vs free	Rate limiter initialized; two clients with different tiers	1. Free client makes 2 requests/second 2. Premium client makes 51 requests/second	Free client blocked after 1st; Premium client blocked after 50th request	Tier-based limits enforced correctly
TC-RL-06	Calculate retryAfter header correctly	Free tier client at per-second limit	1. Exceed per-second limit 2. Check retryAfter value in response	429 response with retryAfter ≤ 1 second	Retry-After header provides accurate wait time
TC-RL-07	Isolate rate limits per client	Rate limiter tracking multiple clients	1. Client A exceeds limit 2. Client B makes request	Client A blocked; Client B allowed	Per-client isolation maintained
TC-RL-08	Allow requests after time window expires	Free tier client at per-second limit	1. Make 1 request 2. Wait 1.1 seconds	First request allowed; After wait	second request also allowed

			3. Make another request		
TC-RL-09	Track requests in all windows simultaneously	Rate limiter initialized	1. Make 3 requests 2. Check statistics for client	All windows (second/minute/hour/day) show count = 3	Concurrent window tracking accurate
TC-RL-10	Reset client rate limits via admin API	Rate limiter with client at limit	1. Client exceeds limit 2. POST /admin/rate-limits/:client/reset 3. Client makes new request	Client blocked before reset; Allowed after reset	Reset operation successful
TC-API-01	Validate correct API key	API key manager loaded with test clients; clients.csv contains valid keys	1. Send request with header X-API-Key: test-key-123 2. Validate key	API key validated successfully; Returns clientName and tier	Client identity retrieved
TC-API-02	Reject invalid API key	API key manager initialized	1. Send request with invalid-key-999 2. Validate key	Validation fails; Returns error 'API key not found'	Invalid keys properly rejected
TC-API-03	Reject empty API key	API key manager initialized	1. Send request with empty X-API-Key header 2. Validate key	Returns error 'Invalid API key format'; 401 Unauthorized	Empty keys rejected

TC-API-04	Reject null API key	API key manager initialized	1. Send request without X-API-Key header 2. Validate key	Returns error message; 401 Unauthorized response	Missing API key handled gracefully
TC-API-05	Handle API keys with whitespace	API key manager loaded; valid key with spaces	1. Send X-API-Key: ' test-key-123 ' 2. Validate key	Whitespace trimmed; Key validated successfully	Whitespace handling works
TC-API-06	Reload API keys from file	API key manager initialized; clients.csv modified	1. Initial key count = 5 2. Modify CSV to add new client 3. Call reloadClients() 4. Validate new key	New key validated successfully; Client count updated	Hot-reload functionality works
TC-API-07	Handle duplicate API keys in CSV	API key manager; CSV contains duplicate keys	1. Load clients.csv with duplicate keys 2. Validate duplicate key	Only first occurrence loaded; Duplicate ignored	Duplicate handling prevents conflicts
TC-API-08	Validate all supported tiers	API key manager with clients of all tiers	1. Validate free tier key 2. Validate basic tier key 3. Validate standard tier key 4. Validate premium tier key	All tiers validated correctly with proper classification	All 5 tiers supported

			5. Validate enterprise tier key		
TC-CI-01	Identify client with valid API key and IP	Client identifier initialized; request with valid API key	1. Create mock request with X-API-Key: key123 2. Set req.ip = 192.168.1.1 3. Call identifyClient(req)	Returns valid ClientIdentity with clientName and tier	Client identification successful
TC-CI-02	Track client IP addresses	Client identifier with IP tracking enabled	1. Client makes request from 192.168.1.100 2. Check learned IPs	IP address recorded in client_ips.csv with timestamp	IP learning and tracking works
TC-CI-03	Match IP against CIDR ranges	Client identifier with CIDR rules; 192.168.1.0/24 defined for client	1. Request from 192.168.1.50 2. Identify client	Client identified via CIDR match; Returns client name	CIDR range matching functional
TC-CI-04	Handle missing API key header	Client identifier initialized	1. Send request without X-API-Key header 2. Attempt identification	Returns invalid identity; Error message provided	Missing API key handled gracefully
TC-IP-01	Block request from blocklisted IP	IP blocklist contains 192.168.1.100	1. Request from IP 192.168.1.100 2. Check IP status	Request blocked with 403 Forbidden; Reason: 'IP address is blocklisted'	Blocklist enforcement works

TC-IP-02	Allow request from allowlisted IP	IP allowlist contains 172.16.0.50	1. Request from IP 172.16.0.50 2. Check IP status	IP allowlisted; Request logged and allowed	Allowlist detection works
TC-IP-03	Process IP not in any list	IP manager initialized; IP not in allow/block lists	1. Request from IP 1.2.3.4 2. Check IP status	Returns action=NONE; Process according to normal rules	Unlisted IPs handled correctly
TC-IP-04	Prioritize blocklist over allowlist	IP in both allowlist and blocklist	1. Add 192.168.1.100 to both lists 2. Request from this IP 3. Check status	Request blocked; Blocklist takes precedence	Blocklist priority enforced
TC-IP-05	Match IP in CIDR range - allowlist	Allowlist contains 192.168.1.0/24	1. Request from 192.168.1.50 2. Check allowlist status	IP detected in allowlisted CIDR range; Allowed	CIDR matching in allowlist works
TC-IP-06	Match IP in CIDR range - blocklist	Blocklist contains 10.0.0.0/8	1. Request from 10.0.5.100 2. Check blocklist status	IP detected in blocklisted CIDR range; Blocked	CIDR matching in blocklist works
TC-IP-07	Handle IPv6 addresses	IP manager with IPv6 support	1. Add 2001:db8::1234 to allowlist 2. Request from this IPv6 address	IPv6 address allowlisted; Request allowed	IPv6 support functional

TC-IP-08	Add IP to allowlist via admin API	IP manager initialized	1. POST /admin/allowlist/add 2. Body: {ip: '5.6.7.8', description: 'Test'} 3. Verify addition	IP added successfully; Saved to CSV file	Dynamic allowlist management works
TC-IP-09	Remove IP from blocklist via admin API	IP manager with IPs in blocklist	1. DELETE /admin/blocklist/remove 2. Body: {ip: '192.168.1.100'} 3. Verify removal	IP removed; No longer in blocklist; Saved to CSV	Dynamic blocklist management works
TC-IP-10	Track request counts for listed IPs	IP manager tracking requests	1. Allowlisted IP makes 5 requests 2. Check statistics	Request count = 5 for that IP in allowlist stats	Request counting accurate
TC-POL-01	Match client-specific policy (highest priority)	Policy manager with multiple policies; request with API key	1. Request: api_key=client123, endpoint=/api/users 2. Policies exist for: client-specific, endpoint-specific, global 3. Select best policy	Client-specific policy selected; Score > 10000; Limit applied	Client-specific priority enforced

TC-POL-02	Match endpoint-specific policy	Policy manager initialized; request without API key	1. Request: endpoint=/api/users 2. Policies: endpoint-specific and global tier 3. Select best policy	Endpoint-specific policy selected; Score ~1000	Endpoint-specific priority over global
TC-POL-03	Fall back to global tier policy	Policy manager initialized; request with only tier	1. Request: tier=free (no endpoint/API key match) 2. Only global tier policy exists 3. Select policy	Global tier policy selected; Score = 50	Global tier fallback works
TC-POL-04	Match parameterized endpoint	Policy manager with parameterized endpoints	1. Request: endpoint=/api/users/123 2. Policy: /api/users/:id 3. Select policy	Parameterized endpoint matched; Correct limit applied	Parameter matching functional
TC-POL-05	Match wildcard endpoint	Policy manager with wildcard policy	1. Request: endpoint=/api/anything 2. Policy: /api/* (wildcard) 3. Select policy	Wildcard policy matched; Applied correctly	Wildcard endpoint matching works
TC-POL-06	Match exact IP address	Policy manager with IP-specific policy	1. Request: ip=192.168.1.100 2. Policy for exact IP exists	IP-specific policy selected; Score = 332	Exact IP matching works

			3. Select policy		
TC-POL-07	Match CIDR range in policy	Policy manager with CIDR policy	1. Request: ip=192.168.1.50 2. Policy: 192.168.1.0/24 3. Select policy	CIDR-based policy matched; Applied to request	CIDR-based policies work
TC-POL-08	Resolve conflict with overlapping rules	Policy manager with all rule types	1. Request matches client-specific, endpoint-specific, and global 2. Calculate scores 3. Select highest	Client-specific wins; Deterministic selection	Conflict resolution deterministic
TC-POL-09	Return null when no policies match	Policy manager initialized; unmatched request	1. Request: endpoint=/unknown, tier=unknown 2. No matching policies exist 3. Attempt selection	Returns null; No policy applied	No-match scenario handled
TC-POL-10	Handle case-insensitive tier matching	Policy manager with tier-based policy	1. Policy: tier=premium 2. Request: tier=PREMIUM (uppercase) 3. Match policy	Policy matched; Case ignored in tier comparison	Case-insensitive matching works
TC-HR-01	Detect file changes and trigger reload	Config manager with file watching enabled; rate_limit_polic	1. Modify rate_limit_policies.csv	File change detected; Configuration	Hot-reload detection works

		ies.csv monitored	2. Wait 2 seconds 3. Check reload triggered	reloaded within 5 seconds	
TC-HR-02	Validate configuration before applying	Config manager initialized; invalid CSV prepared	1. Update CSV with invalid data 2. Trigger reload 3. Check status	Validation fails; Old configuration retained; Error logged	Validation prevents bad config
TC-HR-03	Rollback to previous version	Config manager with version history	1. Current version = v5 2. POST /admin/policies/rollback 3. Check active version	Rolled back to v4; Previous configuration active	Rollback functionality works
TC-HR-04	Maintain version history (last 10)	Config manager; make 12 configuration changes	1. Make 12 successive reloads 2. Check version history 3. Verify count	Only last 10 versions retained; Oldest discarded	Version history management correct
TC-HR-05	Apply atomic configuration updates	Config manager; simultaneous requests during reload	1. Trigger configuration reload 2. Send requests during reload 3. Verify consistency	All requests use either old or new config; No mixed state	Atomic updates maintained

TC-HR-06	Reload multiple CSV files together	Config manager monitoring multiple files	1. Update clients.csv 2. Update rate_limit_policies.csv 3. Trigger reload	Both files reloaded; All changes applied together	Multi-file reload works
TC-HR-07	Handle reload failures gracefully	Config manager; CSV file deleted during reload	1. Delete CSV file 2. Attempt reload 3. Check system state	Reload fails; Previous configuration retained; System stable	Error handling prevents crashes
TC-HR-08	Track reload statistics	Config manager with statistics tracking	1. Perform 5 successful reloads 2. Perform 2 failed reloads 3. GET /admin/policies/stats	Stats show: total=7	success=5
TC-EM-01	Return custom error message for rate limit	Error message manager initialized; rate limit exceeded	1. Client exceeds rate limit 2. System generates error response 3. Check message	Custom branded error message with retryAfter; Contains {{clientName}} replacement	Custom error messages work
TC-EM-02	Substitute template variables correctly	Error message manager with template	1. Template: 'Client {{clientName}} exceeded {{limit}}'	Message: 'Client TestClient exceeded 100'	Variable substitution accurate

			2. Variables: clientName='TestClient', limit=100 3. Generate message		
TC-EM-03	Handle missing template gracefully	Error message manager; template for 'custom_error' not defined	1. Request error message for 'custom_error' type 2. Check fallback	Returns default generic error message; System doesn't crash	Missing template fallback works
TC-EM-04	Update error message via admin API	Error message manager initialized	1. PUT /admin/error-messages/rate_limit 2. Body: new template 3. Verify update	Message updated; Saved to CSV; Subsequent errors use new template	Dynamic message updates work
TC-EM-05	Reset error message to default	Error message manager; custom message set	1. Custom message active 2. POST /admin/error-messages/rate_limit/reset 3. Check message	Message reset to default template; Custom version discarded	Reset to defaults works
TC-EM-06	Update contact email in all messages	Error message manager with {{contactEmail}} templates	1. PUT /admin/error-messages/contact-email	All messages containing {{contactEmail}} updated with new email	Global email update works

			2. Body: {email: 'support@example.com'} 3. Check messages		
TC-EM-07	Handle multiple variable substitutions	Error message manager with complex template	1. Template with {{var1}}, {{var2}}, {{var3}} 2. Provide all variables 3. Generate message	All variables substituted correctly in order	Multiple substitutions handled
TC-CV-01	Validate correct rate limit configuration	Configuration validator initialized; valid config object	1. Config with all required fields 2. Validate against rateLimitConfig schema 3. Check result	Validation passes; result.valid=true; No errors	Valid config accepted
TC-CV-02	Reject configuration with missing required fields	Configuration validator; config missing 'tiers' field	1. Submit config without 'tiers' 2. Validate 3. Check errors	Validation fails; Error type: MISSING_REQUIRED; Field: 'tiers'	Missing fields detected
TC-CV-03	Reject configuration with negative limits	Configuration validator; config with negative values	1. Config: tier.free.limits.second = -1 2. Validate 3. Check errors	Validation fails; Error type: RANGE_ERROR; Message mentions 'below minimum'	Negative values rejected

TC-CV-04	Detect tier hierarchy violations	Configuration validator; premium tier with lower limits than free	1. Config: free.second=10, premium.second=5 2. Validate semantic rules 3. Check errors	Validation fails; Error type: TIER_HIERARCHY_VIOLATION	Hierarchy validation works
TC-CV-05	Detect time window logic errors	Configuration validator; illogical window limits	1. Config: second=100, minute=50 2. Validate (100*60 > 50) 3. Check errors	Validation fails; Error type: WINDOW_LOGIC_ERROR	Window logic validated
TC-CV-06	Detect duplicate API keys in client config	Configuration validator; clientConfig with duplicate keys	1. Two clients with same API key 2. Validate clientConfig 3. Check errors	Validation fails; Error type: DUPLICATE_API_KEY	Duplicate detection works
TC-CV-07	Detect IP list conflicts	Configuration validator; IP in both allowlist and blocklist	1. IP 192.168.1.100 in both lists 2. Validate ipListConfig 3. Check errors	Validation fails; Error type: IP_LIST_CONFLICT	IP conflict detection works
TC-CV-08	Detect invalid CIDR masks	Configuration validator; invalid CIDR notation	1. Config: ip_or_cidr='192.168.1.0/40' (invalid mask) 2. Validate 3. Check errors	Validation fails; Error type: INVALID_CIDR_MASK	CIDR validation accurate

TC-CV-09	Parse and validate JSON configuration	Configuration validator; JSON string input	1. Submit valid JSON string 2. Parse and validate 3. Check result	JSON parsed successfully; Validation passes	JSON parsing works
TC-CV-10	Detect JSON syntax errors	Configuration validator; malformed JSON	1. Submit '{ invalid json' 2. Parse 3. Check errors	Parsing fails; Error type: SYNTAX_ERROR; Descriptive message	Syntax error detection works
TC-AB-01	Block IP address permanently	Admin block manager initialized	1. POST block: source='1.2.3.4', type='ip', admin='alice' 2. Check if blocked 3. Verify persistence	IP blocked; Returns true; Saved to CSV file	Permanent IP blocking works
TC-AB-02	Block API key temporarily with expiry	Admin block manager; time-based blocking	1. Block API key with expiry=100ms 2. Check immediately: isBlocked() 3. Wait 150ms 4. Check again	Initially blocked; After expiry	no longer blocked
TC-AB-03	Block CIDR range and match IPs within	Admin block manager with CIDR support	1. Block: source='10.0.0.0/24', type='cidr' 2. Check IP 10.0.0.5 3. Check IP 10.0.1.5	10.0.0.5 blocked (in range); 10.0.1.5 not blocked	CIDR-based blocking works

TC-AB-04	Unblock previously blocked source	Admin block manager with blocked IP	1. Block IP 1.2.3.4 2. Verify blocked 3. Unblock IP 1.2.3.4 4. Verify unblocked	Initially blocked; After unblock	returns false
TC-AB-05	Track block audit trail	Admin block manager with audit logging	1. Block IP with reason='suspicious' 2. Unblock IP with reason='resolved' 3. Check audit file	Audit file contains both actions with timestamps and admins	Audit trail maintained
TC-AB-06	List all active blocks	Admin block manager with multiple blocks	1. Block 3 different sources 2. Call listBlocks() 3. Verify count	Returns array with 3 blocks; Contains all sources and types	List operation accurate
TC-AB-07	Handle IPv6 CIDR blocking	Admin block manager; IPv6 support	1. Block: source='2001:db8::/32', type='cidr' 2. Check IP 2001:db8::5 3. Verify blocked	IPv6 IP in CIDR range blocked; Matching works	IPv6 CIDR blocking works
TC-AB-08	Persist blocks across restarts	Admin block manager	1. Create manager	IP still blocked after reload;	Block persistence works

			instance; block IP 2. Destroy instance 3. Create new instance; check IP	Persistence works	
TC-SYS-01	Complete request flow - successful	System running; valid client; within limits	1. Send GET /data with valid API key 2. Check response status 3. Verify headers	200 OK response; Data returned; Rate limit headers present	End-to-end flow successful
TC-SYS-02	Complete request flow - rate limited	System running; client at rate limit	1. Exceed per-second limit for free tier 2. Send additional request 3. Check response	429 Too Many Requests; Error message with retryAfter; Custom branded message	Rate limiting in full flow works
TC-SYS-03	Complete request flow - invalid API key	System running; invalid API key provided	1. Send GET /data with invalid-key-999 2. Check response 3. Verify error	401 Unauthorized; Error message: 'API key not found'	Authentication failure handled
TC-SYS-04	Complete request flow - blocklisted IP	System running; IP in blocklist	1. Send request from blocklisted IP 192.168.1.100	403 Forbidden; Message: 'IP address is blocklisted'; Request	IP blocking in full flow works

			2. Check response	blocked before rate limit check	
TC-SYS-05	Health endpoint returns status	System running	1. Send GET /health 2. Check response	200 OK; Response body: {status: 'healthy'}	Health check functional
TC-SYS-06	Admin endpoint - Get client statistics	System running; client has made requests	1. GET /admin/rate-limits/:clientName 2. Check response	200 OK; Statistics showing request counts per window; Remaining requests	Admin stats endpoint works
TC-SYS-07	Admin endpoint - Update tier limits	System running; admin privileges	1. PUT /admin/rate-limits/tier/free 2. Body: {second: 2, minute: 20} 3. Verify changes	200 OK; Tier limits updated; Subsequent requests use new limits	Dynamic tier updates work
TC-SYS-08	Admin endpoint - Reset client rate limits	System running; client at limit	1. Client blocked by rate limit 2. POST /admin/rate-limits/:client/reset 3. Client makes new request	Reset successful; Client allowed to make requests again	Admin reset in full flow works
TC-SYS-09	Multiple clients concurrent requests	System running; 3 clients making simultaneous requests	1. Client A, B, C send requests at same time 2. Check all responses	All clients processed independently; Limits isolated per client; No interference	Concurrent handling works

			3. Verify isolation		
TC-SYS-10	Request with rate limit headers	System running; client makes request	1. Send valid request 2. Check response headers 3. Verify headers present	Response contains: X-RateLimit-Limit	X-RateLimit-Remaining
TC-INT-01	Policy hierarchy with multiple matching rules	Policy manager integration test; multiple policy levels	1. Create client-specific, endpoint-specific, and global policies 2. Send request matching all three 3. Verify selected policy	Client-specific policy selected with highest score; Correct limit enforced	Hierarchy integration works
TC-INT-02	Hot-reload propagation to rate limiter	Integration of config manager and rate limiter	1. Initial limits: free.second=1 2. Update config: free.second=5 3. Trigger reload 4. Test with 3 requests/second	First 3 requests allowed after reload; New limits active	Hot-reload propagation works
TC-INT-03	Client identification with IP tracking	Integration of client identifier and IP manager	1. Client makes first request 2. Check client_ips.csv	IP recorded after first request; Second request recognizes IP;	IP tracking integration works

			3. Client makes second request from same IP	seen_count incremented	
TC-INT-04	Policy validation before reload	Integration of validator and config manager	1. Prepare invalid policy CSV 2. Attempt reload via /admin/policies/reload 3. Check configuration version	Reload rejected; Validation errors returned; Old config still active	Validation integration works
TC-INT-05	Error message customization in rate limit flow	Integration of error message manager and rate limiter	1. Configure custom rate limit message 2. Exceed rate limit 3. Check error response	Custom error message in response; Variables substituted; Branded content	Error message integration works
TC-PERF-01	Handle 1000 requests per client	Performance test; single client	1. Client sends 1000 sequential requests 2. Track response times 3. Verify limits enforced	All requests processed; Average response time < 5ms; Limits enforced correctly	Performance acceptable for high load
TC-PERF-02	Handle 100 concurrent clients	Performance test; multiple clients simultaneously	1. 100 clients each send 10 requests concurrently 2. Track throughput	All 1000 requests processed; No errors; Per-client limits maintained	Concurrent client handling scales

			3. Verify correctness		
TC-PERF-03	Configuration reload under load	Performance test; reload during active traffic	1. Start continuous request flow (100 req/sec) 2. Trigger config reload 3. Verify no request failures	Reload completes in < 50ms; Zero requests failed; Seamless transition	Reload doesn't impact availability
TC-PERF-04	Large configuration file handling	Performance test; 1000 policies loaded	1. Load CSV with 1000 policies 2. Make request matching last policy 3. Track selection time	Policy selected correctly; Selection time < 10ms	Large configs handled efficiently
TC-EDGE-01	Handle empty API key header	Edge case; empty string in header	1. Send X-API-Key: " 2. Attempt authentication	401 Unauthorized; Error: 'Invalid API key format'	Empty values rejected
TC-EDGE-02	Handle malformed CIDR notation	Edge case; invalid CIDR in policy	1. Policy with IP: '192.168.1.0/999' 2. Load configuration 3. Attempt matching	Treated as literal string; No CIDR matching applied; System doesn't crash	Malformed CIDR handled gracefully
TC-EDGE-03	Handle very large rate limits	Edge case; extremely high limit values	1. Configure tier with	Validation warning generated;	High limits handled without overflow

			second=10000 00 2. Validate configuration 3. Apply limits	Limits applied; System functions normally	
TC-EDGE-04	Handle request without IP address	Edge case; req.ip is null/undefined	1. Mock request with no IP 2. Attempt client identification 3. Check result	System handles gracefully; Continues with other identification methods	Missing IP doesn't crash system
TC-EDGE-05	Handle simultaneous reload requests	Edge case; multiple admin reload calls	1. Trigger 3 simultaneous /admin/policies/reload calls 2. Monitor reload process 3. Verify consistency	Reloads serialized or deduplicated; Final state consistent; No race conditions	Concurrent reloads handled safely