# Statement of Work: Traffic Rate-Limiting Implementation for Open VSX

## Objective

The primary goal of this engagement is to design and implement traffic rate-limiting for the Eclipse Foundation–hosted Open VSX service. This functionality will ensure fair usage, prevent abuse, and provide long-term sustainability of the registry.  The contractor will design and implement mechanisms for global request limits, configurable overrides, tiered API limits, detailed logging, and clear feedback messages to clients when limits are exceeded.

## Background

Eclipse Open VSX is an open source registry for Visual Studio Code extensions and compatible editors. The Eclipse Foundation operates the public instance at open-vsx.org, which is widely used by developers, IDE vendors, and tool providers as a trusted alternative to the Visual Studio Marketplace.

As usage continues to grow,  rate-limiting has become critical to ensure the long-term sustainability and efficient use of its extension registry. This initiative will introduce robust mechanisms to manage access, outlined in this Statement of Work (SOW), which details the requirements, technical approach, deliverables, timeline, and acceptance criteria for this positive enhancement.

## Scope

1. **Configurable limits and exceptions (admin area):** Provide an administrative interface to define, assign, and manage all rate-limiting policies.
    - **Global defaults:** define baseline quotas for all traffic.
        - **Anonymous users:** Apply a default hourly request limit per IP address to mitigate abuse from bots.
        - **Authenticated users:** apply a higher default limit for logged-in users who are not associated with a tier or override.
        - Admins must be able to configure both values.
    - **Tiered usage levels:** define quotas by tier (e.g., Individual, Corporate, Enterprise).
        - Admins must be able to assign a predefined tier to a user or an organization.
    - **Token management:**
        - **Rate-limit tokens:** admins must be able to create, rotate, and revoke rate-limit tokens for either **individuals or organizations**. These tokens are used only for rate-limiting authentication and do not grant publishing rights. All actions must be logged for audit.

        - **Access tokens:** continue to be generated and managed solely by individual users for publishing. Admins cannot view or manage these tokens. Rate-limiting must recognize them as tied to the user, and apply that user's assigned tier or overrides rate-limit when such tokens are used.
    - **Overrides:** allow an admin to define custom limit overrides for a user or an organization.
        - Each override must include an admin label, internal description, alternate hourly limit, and an optional expiry date, after which the limit reverts to the global or tiered default.
        - **Forward proxy support:** admins must be able to register specific IPs or IP ranges for enterprise deployments.
            - **CIDR format:** IP entries must use CIDR notation.
            - **Multiple entries:** must support multiple CIDRs per override, either comma-separated or one per line.
    - **Rate-limit admin list page:** the admin area must display a list of all configured rate-limit policies and enable admin to create, edit and delete policies.
    - **Rate-limits only apply to API requests**: Login and user profile pages and authentication flows must always remain accessible and excluded from rate-limiting to ensure users can sign in regardless of their network conditions and continue to manage their access tokens (e.g., VPN usage).
2. **Identity-based rate limiting (tokens + authentication):** Enforce quotas at runtime based on client identity.
    - **Rate-limit tokens:** identify either a **user or an organization** and are assigned a tiered or override rate limit. They do not grant publishing rights.
    - **Access tokens:** identify an individual user. When used, the system resolves the owning user and applies the appropriate authenticated default, tier or overrides.
    - **Authenticated sessions:** these are logged-in users via GitHub OAuth (OIDC provider for Open VSX) which use secure session cookies. If not tied to a tier or custom override, they fall under the authenticated global default.
    - **Anonymous users:** fall under the anonymous global default.
    - **Precedence:**
        - Custom overrides.
        - Tiered rate limits.
        - Access Token.
        - Authenticated session.
        - Global anonymous default.
    - **Precedence handling:**
        - A request may include an access token (Authorization: Bearer <token>) and/or a Rate-Limit Token (X-OpenVSX-RateLimit-Token: <token>).
            - If both are provided, the **Rate-Limit Token takes precedence**.
        - Explicit overrides always supersede defaults or tier-based limits.
3. **Limit-breach error responses:** Provide clear, user-facing explanations and calls to action when requests are denied.

    - APIs: must return **429 Too Many Requests** with standard headers (X-RateLimit-Limit, X-RateLimit-Remaining, X-RateLimit-Reset, Retry-After) and include a clear call-to-action message in the response body.
    - **Web:** must show a custom error page with a call-to-action message.
    - **Calls to action:** suggest becoming an individual sponsor, joining a Working Group, or exploring Eclipse Foundation membership.
4. **Monitoring and observability:** Provide metrics, logging, and dashboards to support transparency and tuning.
    - Emit metrics to **Prometheus** and provide **Grafana** dashboards for real-time monitoring.
    - Log hourly access counts per IP and per override entry, and record limit-breach events (identity, source, timestamp).
    - Retain logs for up to **365 days** to support audits, analysis and tuning.
5. **Documentation & transition:** Deliver clear documentation and training for long-term operation.
    - Admin guide for configuring global limits, tiers, overrides, tokens, and CIDR lists.
    - API documentation for identity/credential handling and rate-limit headers.
    - Runbooks for monitoring and responding to limit breaches.
    - Knowledge-transfer session with Foundation staff.
    - Future recommendations list for long-term improvements.

# Deliverables

1. **Design specification**
    - Architecture overview, microservices, API, databases needed.
    - Data models for counters, overrides, and logs.
    - Constraints, assumptions, and security considerations.
2. **Implementation artifacts**
    - Spring Boot filter or gateway module for rate limiting.
    - Redis (or in-memory) counter service with expiry management.
    - Admin UI for configuring global defaults and rate-limit policies.
    - API endpoints for rate-limit policies and token lifecycle management.
3. **Rate-limit responses**
    - Error handling across APIs and web pages.
    - 429 Too Many Requests responses with standard headers and call to action in body (X-RateLimit-Limit, X-RateLimit-Remaining, X-RateLimit-Reset, Retry-After).
    - Custom error page for web requests.
4. **Testing suite**
    - Unit tests covering limit enforcement logic.
    - Integration tests simulating spikes, overrides, and expiry scenarios.
    - Load tests validating performance under target throughput.
5. **Documentation & knowledge transfer**
    - Configuration guides for global limits and overrides.
    - API docs for token-based and authentication-based rate limiting.

    - Release notes and user communications templates for limit-exceeded feedback.
6. **Monitoring and observability**
    - Prometheus metrics for request counts and override activity.
    - Grafana dashboards for real-time monitoring and anomaly detection.
    - Logging of daily access counts per IP/override and breach events, retained up to 365 days.
7. **Deployment scripts (as applicable)**
    - Helm charts or Docker Compose adjustments.
    - CI/CD pipeline updates for new configuration parameters.

All Deliverables must be contributed to the Eclipse Open VSX project and related resources under the project's license. If the contractor is not currently a Committer on the project, then working closely with an existing Committer to ensure the Deliverables are accepted is considered part of the work effort.

All work must be done in conformance with the Eclipse Development Process. All issues are to be tracked either in public issues, or where directed by Eclipse, to be tracked using confidential issues and resources.

## Skills Required

- Strong proficiency in **Java (JDK 17+)** and **Spring Boot 3.x** for backend services (core of the Open VSX server).
- Solid experience with **TypeScript** and **React** for building and maintaining the admin UI.
- Expertise with **Redis** for rate-limiting counters, caching, and TTL-based enforcement.
- Proficiency in **PostgreSQL** for persistence of tiers, overrides, tokens, and audit logs.
- Practical knowledge of **Docker** and **Kubernetes** for deployment and scaling.
- Experience with **CI/CD pipelines** (GitHub Actions, Jenkins) for automated builds, testing, and deployment.
- Strong understanding of **security best practices**: token hashing/rotation, authentication workflows, and dependency management.
- Experience with stress and load testing to validate rate-limiting scalability and system resilience under peak traffic conditions.
- Knowledge of **rate-limiting algorithms** (e.g., token bucket, leaky bucket, fixed window, sliding window).
- Hands-on experience with monitoring and observability using **Prometheus** and **Grafana**, including dashboards and alerting.
- Comfort working in **open source** contribution workflows under the **Eclipse Foundation process**.

## Possible Technical Approach

- Leverage Spring Boot's filter chain to intercept and evaluate each request.
- Use Redis sorted sets or atomic counters with TTL to aggregate hourly counts.
- Store override configurations in a relational table and cache them for fast lookup.

- Emit metrics to Prometheus and expose Grafana dashboards for real-time monitoring.
- Implement idempotent scripts to migrate existing configurations and seed initial tiers.

These are possible technical approaches. Contractors are encouraged to propose improvements or alternative implementations that meet the requirements while ensuring scalability, security, and maintainability.

## Roles & Responsibilities

- **Development Team (Contractor)**: Implement rate-limiting modules, admin UI, and automated tests (unit, integration, and load).
- **DevOps/Infrastructure (EF)**: Provision Redis cluster, configure Prometheus/Grafana, update CD pipelines.
- **Quality Assurance (EF)**: Validate functional correctness, performance, confirm logging, metrics, error responses, and security.
- **Open VSX Admins (EF)**: Define override entries, monitor dashboards, and adjust tier definitions.
- **Project Manager**: Track progress, coordinate reviews, and liaise with stakeholders, elaborate and execute a communication plan.

## Assumptions & Dependencies

- Redis instance or equivalent is available for counter storage.
- Existing OAuth infrastructure can support additional rate-limit metadata.
- The Eclipse Foundation maintains authority to assign override entries and tokens.
- No regulatory or privacy constraints hinder per-IP logging.
- The list of supporters/backers and rate-limit policies must be persistent; this information is expected to be stored in the PostgreSQL database

## Reporting & Communication

- Weekly status reports including progress updates (burn-down charts or equivalent).
- Regular updates to issues on GitHub
- Stakeholder demos at key milestones to showcase dashboards and rate-limit functionality.

## Acceptance Criteria

- Global defaults, tiered usage levels, token management, and overrides are fully implemented and configurable in the admin area.
- Rate limits are enforced correctly under unit, integration, and load tests.
- Override entries apply and expire as expected.
- Metrics and logs capture daily counts and limit-reached events.
- User-facing limit-exceeded messages include rationale and calls to action.
- Admin guides, API docs, runbooks, and training/knowledge-transfer sessions are delivered.

Successful proposals must show how their implementation plan will deliver on each of these acceptance points.

## Proposal Evaluation Criteria

In evaluating proposals, the Eclipse Foundation will consider:

- Price and timeliness,
- Plan for proposed development and deliverables,
- Skillset and experience of proposed developers, with preference given to open source project committers in the relevant area,
- Bidder's relationship with EF, with preference given to either Contributing Members with committers, or self-employed committers with relevant expertise,
- Any additional relevant elements in the bid, including delivery date, whether fixed price vs. time and materials basis, etc.

## Proposal Submission

Proposals must be submitted by email to software-dev@eclipse-foundation.org no later than 18 September 2025 at 18:00 CEST (UTC+2).

Submissions should include:

- A detailed plan addressing the scope, deliverables, and a timeline.
- Profiles of the proposed developers and their relevant experience.
- Pricing information.
- Any additional information the bidder believes is relevant for the proposal.

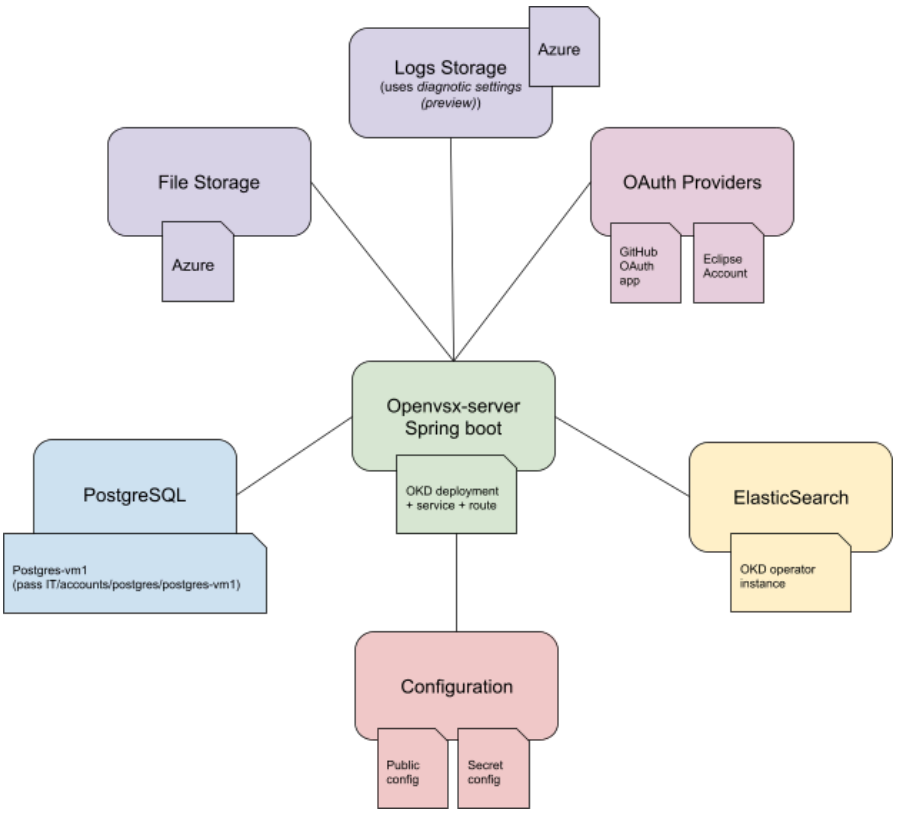
Late submissions may not be considered.

## Duration

Short-term engagement (**[X weeks/months]**, to be defined), focused on implementing traffic rate-limiting. The goal is to deliver a production-ready solution with immediate impact on system sustainability and reliability.

## Architecture & projects

- Eclipse Open VSX project: https://github.com/eclipse/openvsx
- The source of open-vsx.org (the public instance of Eclipse Open VSX) is at: https://github.com/EclipseFdn/open-vsx.org
- Reference documentation on the instance of Open VSX hosted by the Eclipse Foundation is available at: https://github.com/EclipseFdn/open-vsx.org/wiki/Deployment-Details




# References

1. GitHub Issue:
    a. [Provide a means to account for, and rate limit server calls]