Rate Limiting using Godspeed Framework

PRD Document

Name: Aashi Gupta
Date: 25/June/2025
College: IIT Kanpur
Mobile No: 8920813195
Email id: aashigupta24@iitk.ac.in, aashifamily123@gmail.com

1. Abstract

This project is a proof-of-concept (PoC) implementation of a rate-limiting SDK built using the Godspeed Framework for the backend and React.js for the frontend. It functions as a smart API gateway that enforces service-specific quotas and rule-based access control. The system includes features like real-time context evaluation, rule caching, JWT-based secure authentication, and tier-based rate decisions. The SDK ensures high accuracy, performance and includes an admin panel for rule management.

2. Objective

- Implement rate limiting using Godspeed that accepts JWT tokens
- Manage rule fetching, caching, and evaluation
- Handle real-time context data per request
- Provide an admin login panel
- Ensure secure token-based access
- Integrate frontend and backend seamlessly

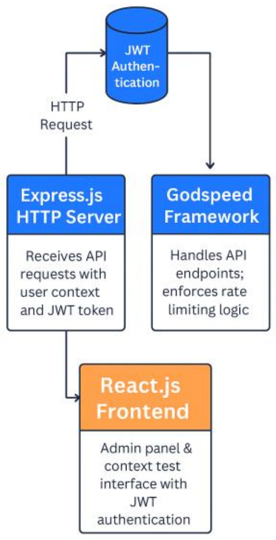
3. Introduction

Rate limiting helps control the volume of traffic to APIs, ensuring fairness and preventing abuse. The Godspeed Framework is ideal for this due to its event-driven, modular, and real-time capabilities.

4. Tech Stack

Layer	**Technology / Tools**
Backend Framework	Godspeed Framework (TypeScript)
Language	TypeScript
API Server	Express.js (configured via http.yaml in Godspeed)
Authentication	JWT (via Godspeed's jwt config)
Authorization	Custom Godspeed policy com.gs.is_allowed
Functions	Modular handlers (fetchContext.ts, fetchRules.ts, evaluateRules.ts etc)
Context Mgmt.	Real-time data fetch per request (userId, orgId, service)
Rule Engine	YAML-based rule definitions (quota, user tier, peak hours)
API Docs	Swagger/OpenAPI auto-generated at /docs
Build Tool	SWC (@swc/core, @swc/cli) for fast TS transpilation
Dev Utilities	nodemon, run-script-os, dotenv, live reload setup
Package Manager	npm
Frontend Framework	React.js (admin panel + context test interface)
Frontend Features	JWT auth, secure API calls, rule config interface
API Integration	Axios/Fetch to call Godspeed REST endpoints
Testing	Swagger UI with token support

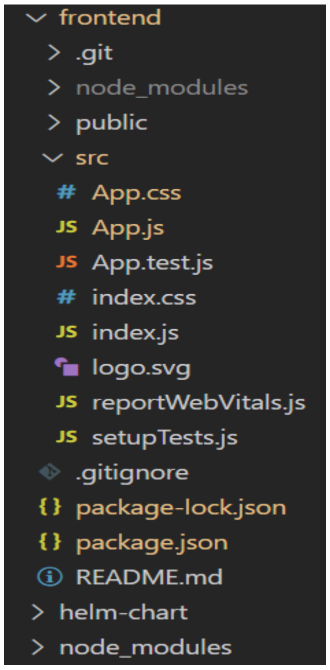
5. Architecture Diagram


JWT Authen-tication
HTTP Request
Express.js HTTP Server
Receives API requests with user context and JWT token
Godspeed Framework
Handles API endpoints; enforces rate limiting logic
React.js Frontend
Admin panel & context test interface with JWT authentication

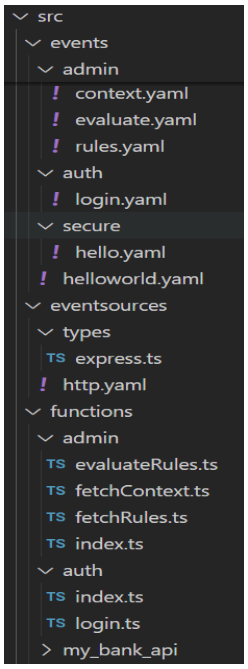
6. Implementation Steps

1) **Project Initialization**

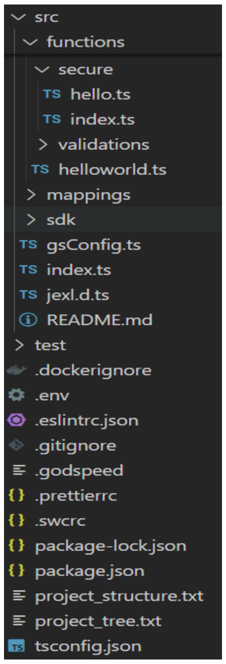
- Install Godspeed: npm install -g @godspeedsystems/godspeed
- Create project: godspeed create my-project
- Setup folders: src/functions, src/events, src/eventsources, etc.
- Setup file structure (src/functions, src/events, src/eventsources, etc.)



```
frontend
  > .git
  > node_modules
  > public
  src
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
    reportWebVitals.js
    setupTests.js
  .gitignore
  package-lock.json
  package.json
  README.md
> helm-chart
> node_modules
```



```
src
  events
    admin
      context.yaml
      evaluate.yaml
      rules.yaml
    auth
      login.yaml
    secure
      hello.yaml
    helloworld.yaml
  eventsources
    types
      express.ts
    http.yaml
  functions
    admin
      evaluateRules.ts
      fetchContext.ts
      fetchRules.ts
      index.ts
    auth
      index.ts
      login.ts
    > my_bank_api
```



```
src
  functions
    secure
      hello.ts
      index.ts
    > validations
    helloworld.ts
  > mappings
  > sdk
  gsConfig.ts
  index.ts
  jexl.d.ts
  README.md
> test
.dockerignore
.env
.eslintrc.json
.gitignore
.godspeed
.prettierrc
.swcrc
package-lock.json
package.json
project_structure.txt
project_tree.txt
tsconfig.json
```

2) **Define HTTP Event Source**

- Edit src/eventsources/http.yaml
- Configure express, Swagger endpoint (/api-docs), JWT, CORS, ports

3) **Create Functions**

• Implement logic in src/functions/admin/

- fetchContext.ts
- fetchRules.ts
- evaluateRules.ts
- index.ts

4) **Create Event Routes**

• Define YAML routes in src/events/

- auth/login.yaml
- admin/context.yaml
- admin/evaluate.yaml
- admin/rules.yaml
- secure/hello.yaml
- helloworld.yaml

5) **PoC Partially implement Context & Rule Engine Logic**

- Simulate context & rule fetching (initially without in-memory caching)
- Extract context data from query parameters

6) **Implement Middleware & Validations**

- Add request/response YAML validation
- Setup fallback error handlers

7) **Build & Serve**

- Backend: godspeed serve
- Frontend: cd frontend && npm start

7. Frontend (React App)

1. **Scaffold App**

- Built with Vite + TypeScript

2. **JWT Integration**

- JWT authentication using jsonwebtoken package

3. **<u>Simulated SDK Features</u>**

- Token issuance and injection
- Fetching rule/context data via secure endpoints

4. **<u>Admin Panel Functionalities</u>**

- Admin login
- JWT token display
- Swagger API integration
- Real-time rule testing interface

8. Demo Flow

1. Admin logs in via React frontend and gets a JWT
2. JWT is used to access secure endpoints (e.g., /secure/hello)
3. Admin tests rules by calling /admin/evaluate with user/org/service
4. Frontend displays the decision (ALLOW/DENY)
5. Rule logic can be tested live

9. API Testing + Swagger UI

- Swagger UI: http://localhost:3100/docs
- React App: http://localhost:3000
- Network testing: http://192.168.29.233:3000

10. Feature Completion Summary

- Secure JWT-based login & token auth
- Protected backend endpoints
- Working rule evaluation at /admin/evaluate
- Swagger docs for all APIs
- React frontend with testing interface
- Full backend-frontend integration

11. Full Project Completion Plan

- Convert SDK to reusable Go/Node.js module
- Rule caching with TTL and auto-refresh
- Real-time context fetch logic
- HTTP 429 error handling
- YAML-based rule builder UI
- Persistent rule storage (PostgreSQL/MongoDB)
- Rule simulation & conflict resolution

- TLS-secured data exchange
- Fallback handling (fail-open/fail-closed)
- SDK packaging (npm/go module)
- Analytics dashboard (traffic, rule hits)

12. Summary of POC Progress

Completed:

- JWT-based backend authentication
- React UI with login and evaluation
- Call to /admin/evaluate functional
- Swagger docs secured
- Basic YAML rule engine in backend

In Progress:

- Dynamic rule fetching per service/org
- UI to display rule JSON data

Planned:

- SDK implementation
- Persistent backend rule storage
- Enhanced admin UI with templates and simulations

13. Testing Information (for Reviewers)

Use the following credentials and data for testing:

- Admin Username: aashi
- Password: aash123
- Sample Org ID: org123
- Sample Service: payment-service

GitHub Repo Link:

https://github.com/ashigupta99/Godspeed_Rate_Lim_SDK

(also has POC execution video in Github Repo)