# Proof of Concept — Spring Boot REST + GraphQL API (Java 17)

This document is a detailed Proof of Concept (PoC) for setting up, running, and testing the Spring Boot REST + GraphQL API project.

## 0) Overview

* REST API with validation, error handling, and API versioning (v1/v2)
* GraphQL endpoint
* Rate limiting via Bucket4j
* Request logging
* In-memory repository for demonstration
* Project compatible with Java 17 and Spring Boot 3.x

## 1) Prerequisites

* Java 17 JDK installed (java -version)
* Maven 3.6+ (mvn -v) or use project wrapper
* Docker (optional, for container run)
* curl, Postman, or GraphiQL for testing

Windows users must set JAVA\_HOME and add %JAVA\_HOME%\bin to PATH.

## 2) Project Layout

springboot-api/  
├─ pom.xml  
├─ src/main/java/com/example/demo/  
│ ├─ DemoApplication.java  
│ ├─ controller/  
│ │ ├─ v1/UserControllerV1.java  
│ │ ├─ v2/UserControllerV2.java  
│ │ └─ UserGraphQLController.java  
│ ├─ dto/UserDto.java  
│ ├─ entity/User.java  
│ ├─ repository/InMemoryUserRepository.java  
│ ├─ service/UserService.java  
│ ├─ exception/ApiException.java  
│ ├─ exception/GlobalExceptionHandler.java  
│ ├─ filter/RequestLoggingFilter.java  
│ └─ filter/RateLimitFilter.java  
├─ src/main/resources/application.yml  
├─ src/main/resources/graphql/schema.graphqls  
├─ Dockerfile  
└─ README.md

## 3) Build & Run

**Build:**

mvn clean package

**Run:**

mvn spring-boot:run

Or run jar:

java -jar target/<artifact>-0.0.1-SNAPSHOT.jar

Default port: 8080.

## 4) REST API Endpoints

### v1 Endpoints

* POST /api/v1/users - Create user
* GET /api/v1/users - List all users
* GET /api/v1/users/{id} - Get user by ID

**Example request:**

curl -i -X POST http://localhost:8080/api/v1/users \  
 -H "Content-Type: application/json" \  
 -d '{"name":"John Doe","email":"john.doe@example.com"}'

**Expected response:** 201 Created, JSON body with id, name, email.

**Validation errors:** Returns 400 with validationErrors JSON.

### v2 Endpoints

* /api/v2/users - Accepts additional fields like phone
* Header versioning supported via X-API-VERSION

## 5) GraphQL Endpoint

* URL: /graphql
* Example mutation (matching mutation createUser { createUser(name:..., email:...) { ... } } format):

mutation createUser {  
 createUser(name: "Xyz", email: "xyz@example.com") {  
 id  
 name  
 email  
 }  
}

**Response:**

{  
 "data": {  
 "createUser": {  
 "id": 1,  
 "name": "Xyz",  
 "email": "xyz@example.com"  
 }  
 }  
}

* GraphQL queries also available for fetching users.

## 6) Rate Limiting

* Default: 10 requests/minute globally
* Exceeding limit returns 429 Too Many Requests

Rate limit exceeded. Try again later.

* Can be adjusted in RateLimitFilter.

## 7) Request Logging

* Logs method, URI, status, and duration
* Example log line:

INFO ... POST /api/v1/users -> 201 (35 ms)

* Filter: RequestLoggingFilter

## 8) API Versioning

* URI-based: /api/v1/... and /api/v2/...
* Header-based: X-API-VERSION
* v2 can include new fields (e.g., phone)

## 9) Docker (Optional)

Build & run:

docker build -t springboot-api:latest .  
docker run -p 8080:8080 springboot-api:latest

* Endpoints available at localhost:8080.

## 10) Troubleshooting

* Cannot invoke "UserDto.getName()" because dto is null: Use scalar arguments in mutation and manually build DTO.
* MethodArgumentNotValidException not triggering: Ensure @Valid and validation dependency are included.
* Port conflicts: Change server.port in application.yml.
* Rate limit always 429: Make sure bucket is initialized once as bean.

## 11) Acceptance Checklist

1. App starts successfully.
2. POST /api/v1/users valid request -> 201 + JSON.
3. POST /api/v1/users invalid -> 400 + validationErrors.
4. GET /api/v1/users returns created users.
5. GraphQL createUser works with scalar args.
6. Exceeding 10 requests/min -> 429.
7. Request logging prints info lines.
8. v2 endpoint works with extra fields.

## 12) Next Steps / Improvements

* Add authentication (JWT/roles)
* Replace in-memory store with relational DB
* Per-client rate limiting
* Metrics and tracing
* Integration tests and CI/CD pipeline

## 13) Quick Commands

* Build: mvn clean package
* Run: mvn spring-boot:run or java -jar target/\*.jar
* Docker: docker build -t springboot-api . && docker run -p 8080:8080 springboot-api
* Test GraphQL:

curl -X POST http://localhost:8080/graphql -H "Content-Type: application/json" -d '{"query":"mutation { createUser(name: \"X\", email: \"x@x.com\") { id name email } }"}'

* Test REST create:

curl -X POST http://localhost:8080/api/v1/users -H "Content-Type: application/json" -d '{"name":"John","email":"john@example.com"}'

End of PoC.