

Assignment 4, golang

Daniyar Anuar
Application development in the Golang language
22.11.2024

Executive Summary

This report describes the implementation of security practices, logging, and monitoring in a Golang web application. Key features include JWT-based authentication, CSRF protection, input validation, hashed password storage, and proper error handling. Structured logging and performance metrics tracking with Prometheus ensure the application's reliability and provide real-time insights into its performance.

Table of Contents

- 1. Introduction**
 - 1.1 Overview
 - 1.2 Purpose and Scope
- 2. Security in Go Web Applications**
 - 2.1 Overview of Security Measures
 - 2.2 Input Validation
 - 2.3 Authentication and Authorization
 - 2.4 Data Protection
 - 2.5 CSRF Protection
 - 2.6 Security Headers
 - 2.7 Error Handling
- 3. Logging and Monitoring in Go Web Applications**
 - 3.1 Structured Logging
 - 3.2 Log Levels
 - 3.3 Request Logging
 - 3.4 Monitoring Setup
 - 3.5 Application Metrics
 - 3.6 Alerting
 - 3.7 Log Management
- 4. Conclusion**
- 5. Recommendations**
- 6. References**

Introduction

1.1 Overview

Security, logging, and monitoring are essential for building reliable and robust web applications. Security practices like authentication, data protection, and CSRF prevention safeguard the application and its users. Logging and monitoring, on the other hand, help in tracking errors, performance, and potential anomalies in real-time.

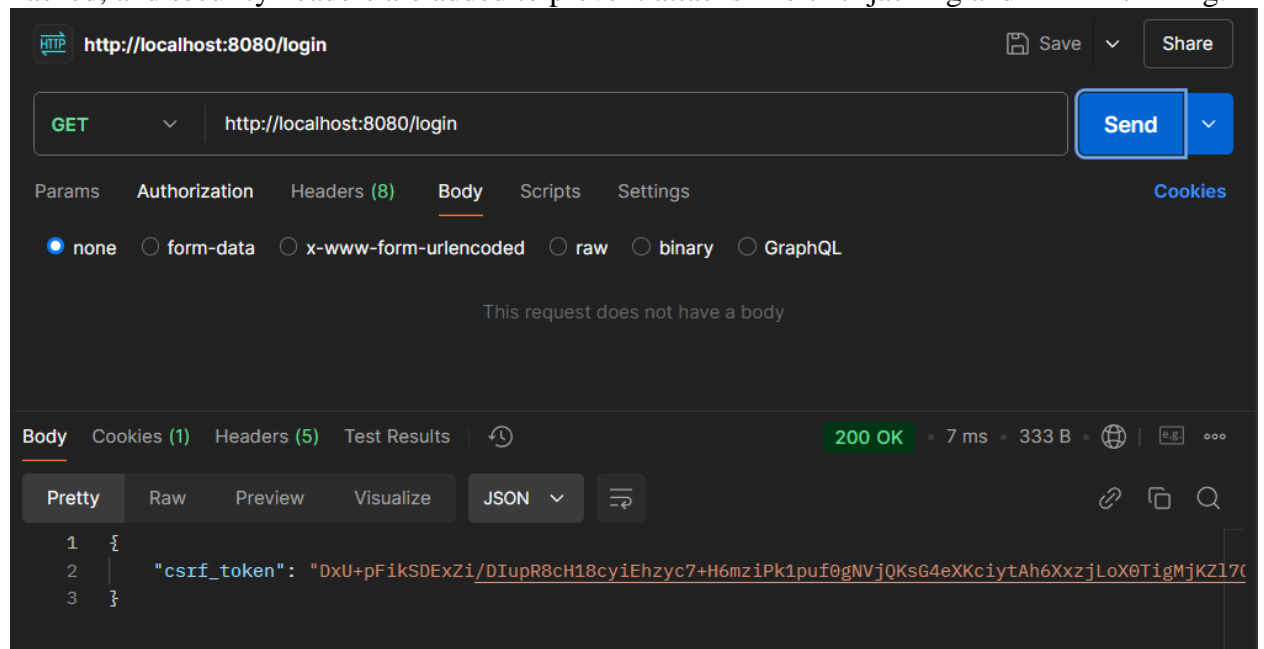
1.2 Purpose and Scope

This report explains how these practices were implemented in a Golang web application using tools like gorilla/mux, gorilla/csrf, and Prometheus. The main focus is on building a secure app with detailed logs and metrics for effective monitoring.

Security in Go Web Applications

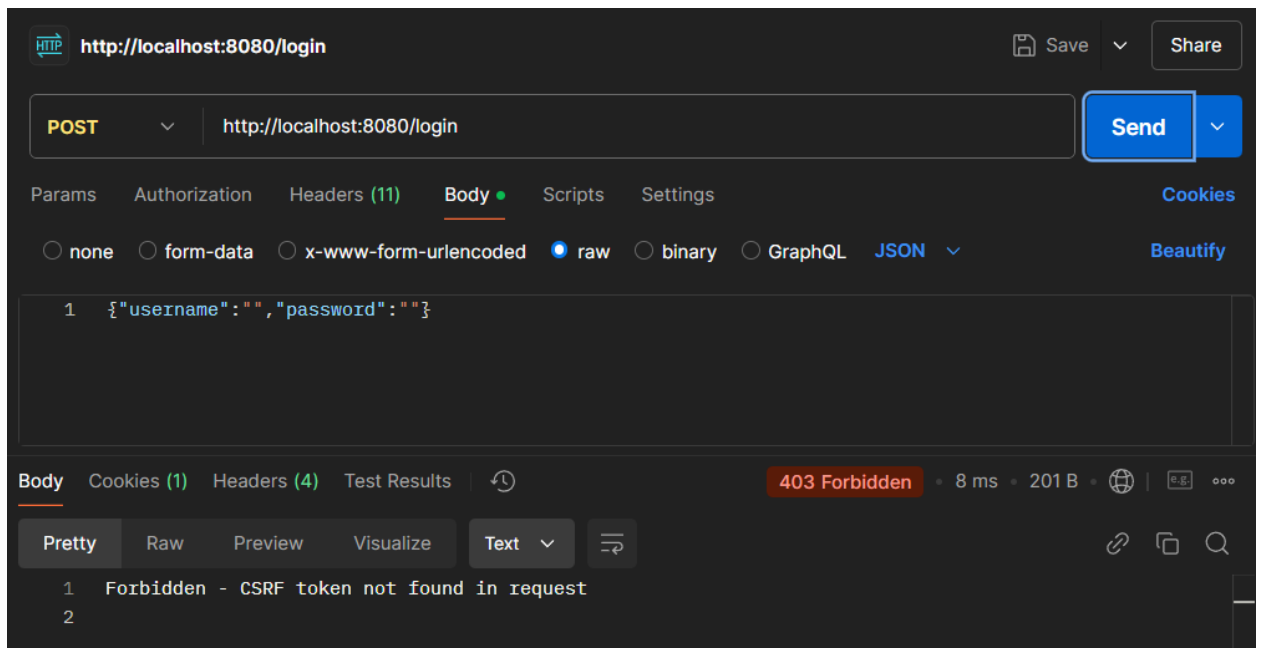
2.1 Overview of Security Measures

The application implements multiple layers of security, including input validation, JWT-based authentication, CSRF protection, and role-based access control (RBAC). Passwords are securely hashed, and security headers are added to prevent attacks like clickjacking and MIME sniffing.



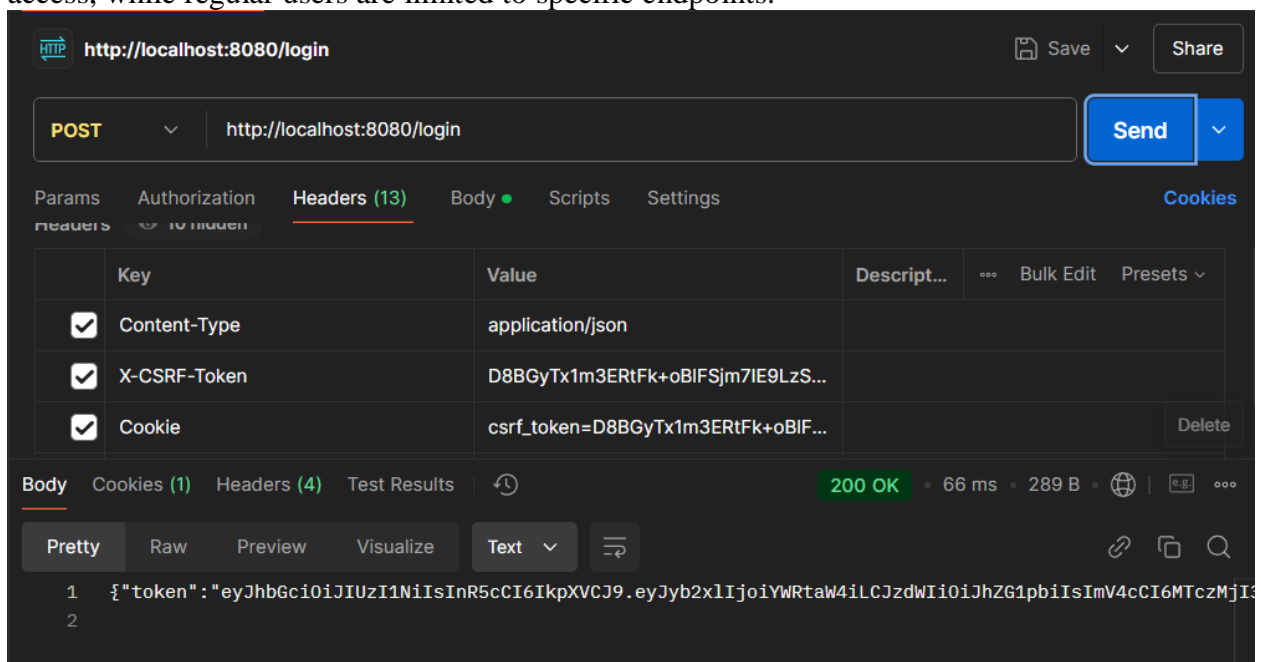
2.2 Input Validation

The go-playground/validator package was used to ensure all user inputs are properly validated. For example, username must be alphanumeric, and password must meet length requirements. Invalid inputs return an error.



2.3 Authentication and Authorization

JWTs are used to authenticate users. After logging in, users receive a token that grants access to protected routes. RBAC ensures users only access what their roles allow. Admins have full access, while regular users are limited to specific endpoints.



2.4 Data Protection

Passwords are hashed using bcrypt before being stored. This ensures that even if the database is compromised, passwords remain secure.

```

func hashPassword(password string) string { 2 usages new *
    hashed, err := bcrypt.GenerateFromPassword([]byte(password), bcrypt.DefaultCost)
    if err != nil {
        log.Fatalf(format: "Error hashing password: %v", err)
    }
    return string(hashed)
}

func checkPasswordHash(password, hash string) bool { 1 usage new *
    err := bcrypt.CompareHashAndPassword([]byte(hash), []byte(password))
    return err == nil
}

```

2.5 CSRF Protection

The application uses gorilla/csrf to prevent CSRF attacks. Each form submission requires a CSRF token, which is validated on the server side.

2.6 Security Headers

Middleware was added to enforce security headers, including:

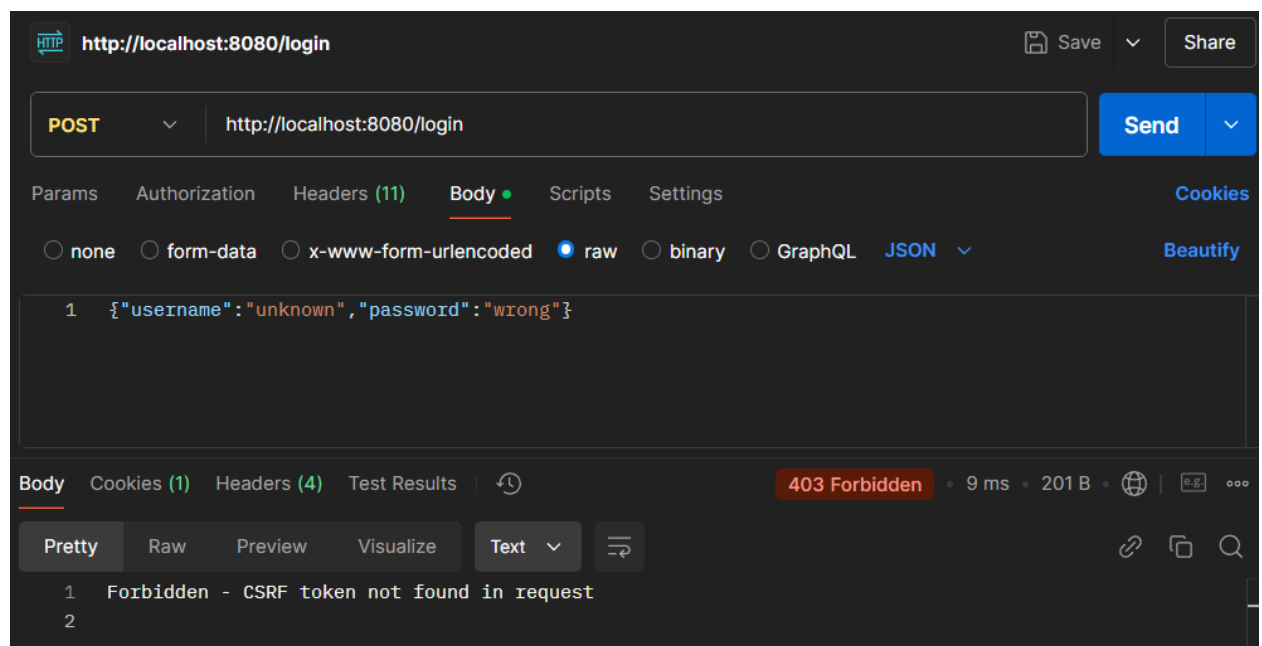
Content-Security-Policy: Prevents loading unauthorized external resources.

X-Frame-Options: Blocks clickjacking attacks by disallowing iframe embedding.

X-Content-Type-Options: Prevents MIME-type sniffing.

2.7 Error Handling

Errors are handled gracefully to avoid exposing sensitive details.



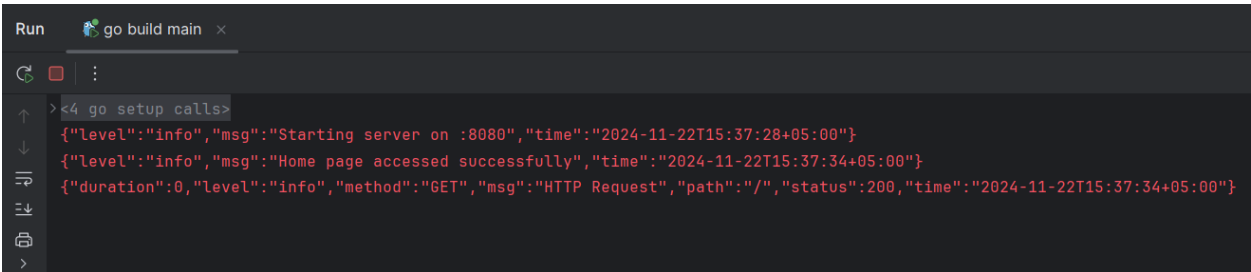
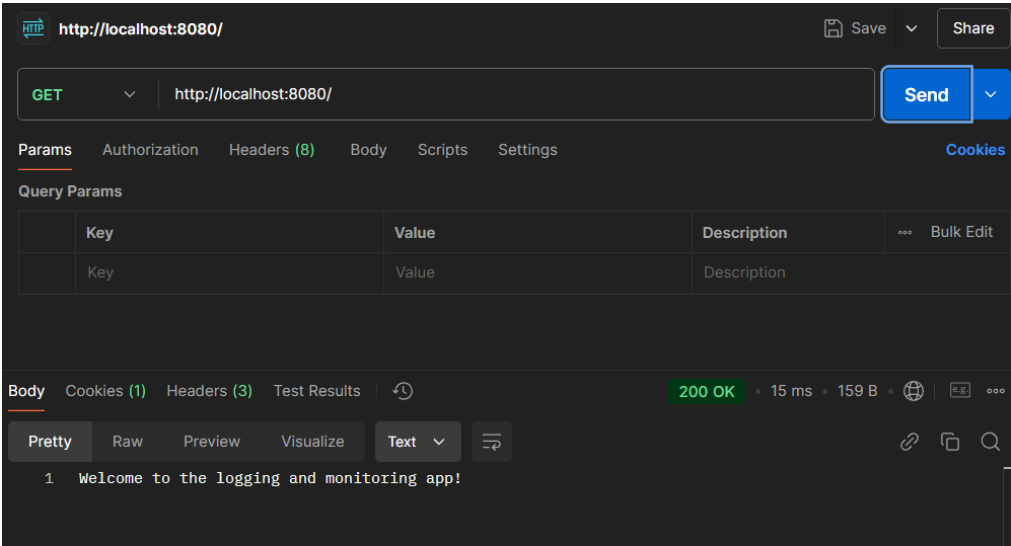
Logging and Monitoring in Go Web Applications

3.1 Overview

The application uses logging and monitoring to improve visibility and ensure reliability. Structured logs capture key events, while Prometheus tracks performance metrics in real time, enabling proactive issue detection.

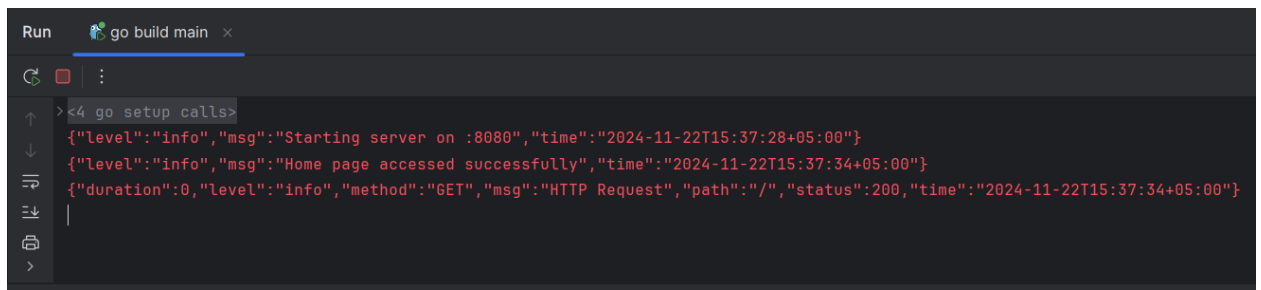
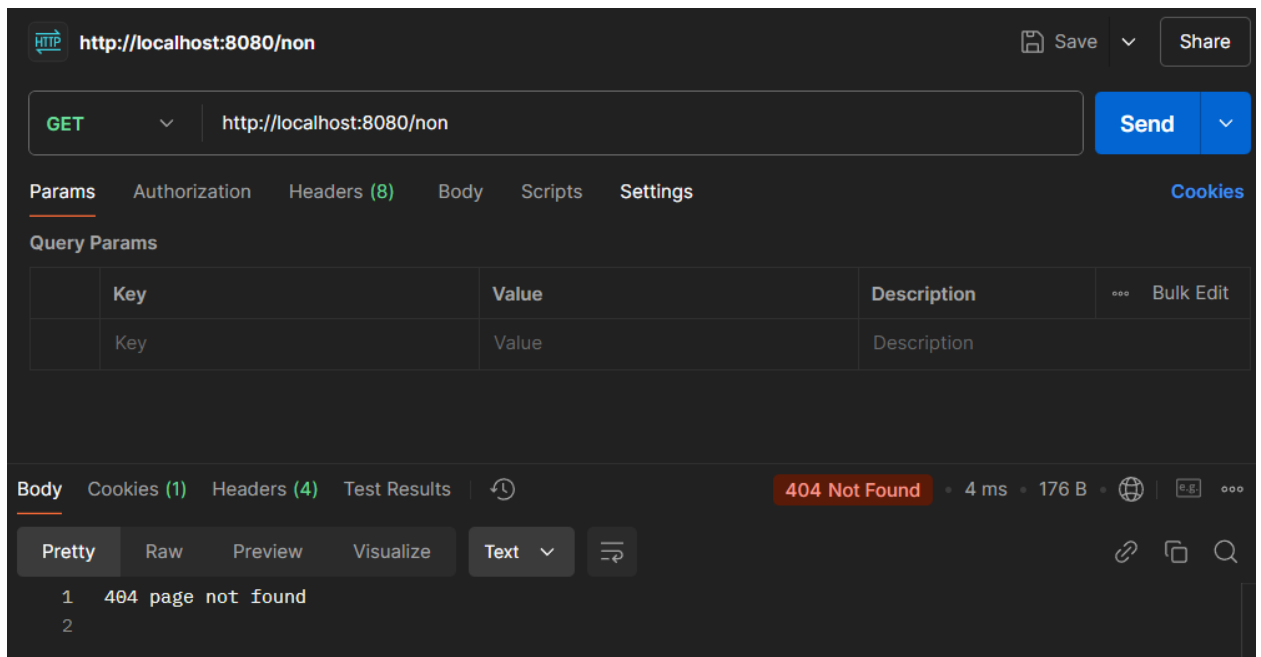
3.2 Structured Logging

Logrus was used for JSON-formatted logs, including details like HTTP method, path, status, and duration. These logs are machine-readable and ideal for analysis.



3.3 Log Levels

- **INFO:** Normal operations (e.g., successful requests).
- **WARN:** Potential problems (e.g., slow responses).
- **ERROR:** Critical issues (e.g., request failures).

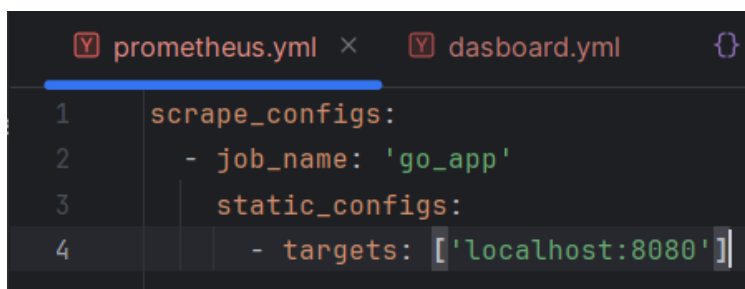


3.4 Request Logging

Middleware logs each HTTP request: method, path, status, and duration. This helps identify slow endpoints and track errors.

3.5 Monitoring Setup

Prometheus collects metrics via /metrics. Example configuration: yml



3.6 Metrics

- **Request Counts:** Tracks total requests by method, path, and status.
- **Response Times:** Measures request duration to detect performance issues.


```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0
go_gc_duration_seconds{quantile="0.75"} 0
go_gc_duration_seconds{quantile="1"} 0
go_gc_duration_seconds_sum 0
go_gc_duration_seconds_count 0
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 7
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.22.5"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 1.049416e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 1.049416e+06
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 1.44922e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 720
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 1.95544e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 1.049416e+06
# HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used.
# TYPE go_memstats_heap_idle_bytes gauge
go_memstats_heap_idle_bytes 1.081344e+06
# HELP go_memstats_heap_inuse_bytes Number of heap bytes that are in use.
# TYPE go_memstats_heap_inuse_bytes gauge
go_memstats_heap_inuse_bytes 2.654208e+06
# HELP go_memstats_heap_objects Number of allocated objects.
# TYPE go_memstats_heap_objects gauge
go_memstats_heap_objects 8090
# HELP go_memstats_heap_released_bytes Number of heap bytes released to OS.
# TYPE go_memstats_heap_released_bytes gauge
go_memstats_heap_released_bytes 1.081344e+06
# HELP go_memstats_heap_sys_bytes Number of heap bytes obtained from system.
# TYPE go_memstats_heap_sys_bytes gauge
go_memstats_heap_sys_bytes 3.735552e+06
# HELP go_memstats_last_gc_time_seconds Number of seconds since 1970 of last garbage collection.
# TYPE go_memstats_last_gc_time_seconds gauge
go_memstats_last_gc_time_seconds 0
# HELP go_memstats_lookups_total Total number of pointer lookups
```

3.7 Alerting

Prometheus alerts notify on:

- **High Error Rate:** More than 5% of requests fail.
- **Slow Requests:** 95% of requests take over 2 seconds.

3.8 Log Management

Logs are rotated using lumberjack to limit size (10 MB), backups (5 files), and retention (30 days).

```
1 {"level":"info","msg":"Starting server on :8080","time":"2024-11-22T15:36:40+05:00"}
2 {"level":"info","msg":"Home page accessed successfully","time":"2024-11-22T15:37:10+05:00"}
3 {"duration":0.0000617,"level":"info","method":"GET","msg":"HTTP Request","path":"/","status":200,"time":"2024-11-22T15:37:10+05:00"}
4 {"level":"info","msg":"Starting server on :8080","time":"2024-11-22T15:38:25+05:00"}
5 {"level":"info","msg":"Starting server on :8080","time":"2024-11-22T15:38:32+05:00"}
6 {"level":"info","msg":"Home page accessed successfully","time":"2024-11-22T15:38:56+05:00"}
7 {"duration":0,"level":"info","method":"GET","msg":"HTTP Request","path":"/","status":200,"time":"2024-11-22T15:38:56+05:00"}
8
```

Conclusion

Security, logging, and monitoring greatly enhance the application's reliability and safety. Security features like input validation, JWT authentication, and CSRF protection safeguard user data, while structured logging and monitoring with Prometheus help identify and resolve issues quickly. Together, these practices ensure the application performs well and remains secure.

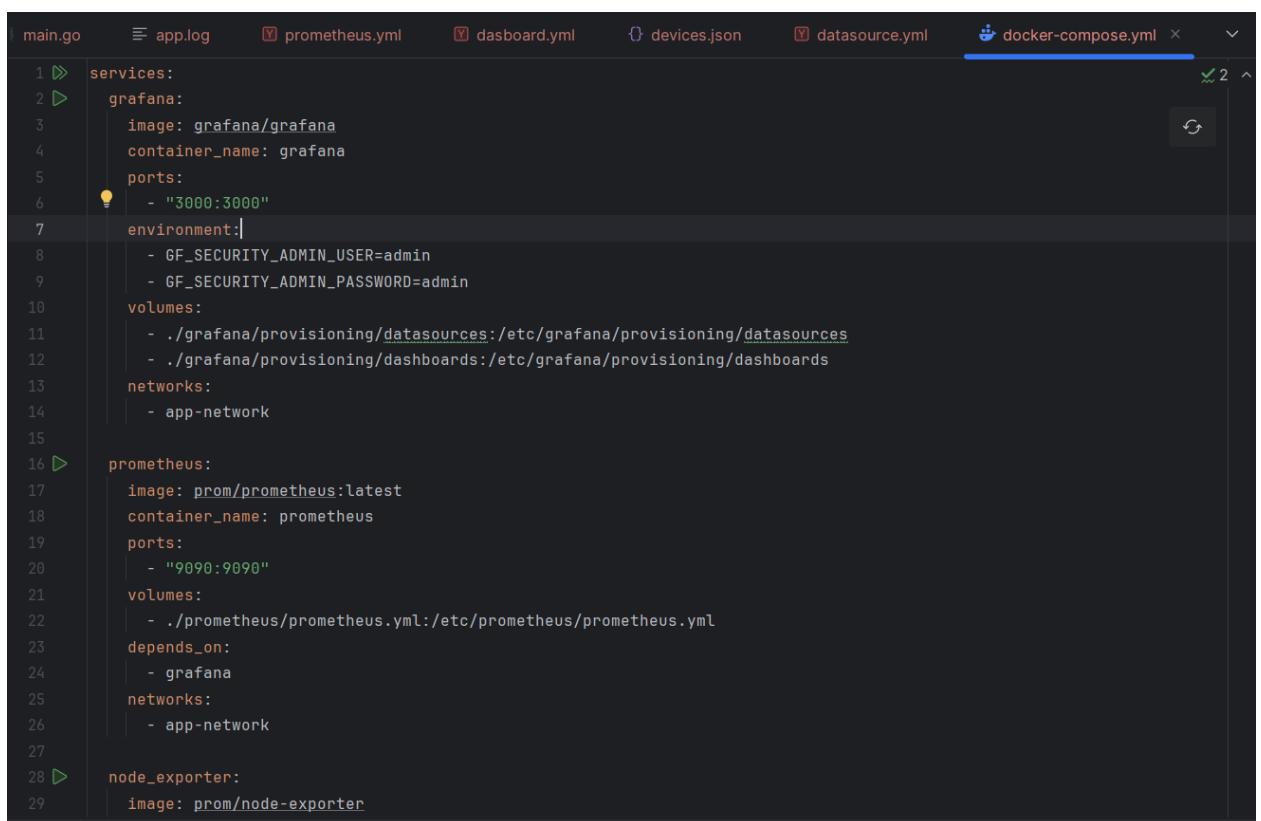
Recommendations

1. Use tools like ELK Stack or Grafana Loki for centralized log analysis.
2. Add more detailed metrics to track application-specific performance.
3. Regularly test and refine alert thresholds to avoid unnecessary noise.
4. Implement HTTPS to secure data in transit.
5. Perform regular security audits and penetration tests.

References

1. Prometheus Docs: prometheus.io
2. Logrus Docs: [logrus](https://github.com/Sirupsen/logrus)
3. Lumberjack Docs: [lumberjack](https://github.com/rabbitmq/lumberjack)
4. Gorilla CSRF Docs: [gorilla/csrf](https://github.com/gorilla/csrf)
5. Go Validator Docs: [go-playground/validator](https://github.com/go-playground/validator)

Appendices



```
1 services:
2   grafana:
3     image: grafana/grafana
4     container_name: grafana
5     ports:
6       - "3000:3000"
7     environment:
8       - GF_SECURITY_ADMIN_USER=admin
9       - GF_SECURITY_ADMIN_PASSWORD=admin
10    volumes:
11      - ./grafana/provisioning/datasources:/etc/grafana/provisioning/datasources
12      - ./grafana/provisioning/dashboards:/etc/grafana/provisioning/dashboards
13    networks:
14      - app-network
15
16  prometheus:
17    image: prom/prometheus:latest
18    container_name: prometheus
19    ports:
20      - "9090:9090"
21    volumes:
22      - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
23    depends_on:
24      - grafana
25    networks:
26      - app-network
27
28  node_exporter:
29    image: prom/node-exporter
```

