

# LOGGING IN GO

# WHAT IS LOGGING?

- Recording events that happen in a program
- Helps in:
  - Debugging errors
  - Monitoring application behavior
  - Auditing and tracing
- Essential for production systems

# WHY LOGGING IS IMPORTANT IN GO

- Go is used for:
  - Web servers
  - Microservices
  - Cloud systems
- Logs help developers:
  - Understand failures
  - Track performance
  - Diagnose crashes

# BUILT-IN LOGGING PACKAGE (LOG)

- Go provides a standard `log` package
- Simple and easy to use
- Features:
  - Print logs with timestamps
  - Output to console or file
  - Fatal and Panic logging

# BASIC LOGGING EXAMPLE

```
import "log"

func main() {
    log.Println("Hello world!")
    log.Fatal("This logs a message then calls os.Exit(1)")
    log.Panic("This logs a message then calls panic()")
    log.Println("This will not work after fatal or panic")
}
```

# LOG LEVELS IN GO

- Common log levels:
  - INFO – general information
  - WARNING – something unusual
  - ERROR – something failed
  - DEBUG – detailed debugging info
- Go's default **log** package does not have built-in levels (need custom or external libraries)

# LOGGING TO A FILE

```
import (
    "log"
    "os"
)

func main() {
    file, err := os.OpenFile("app.log", os.O_APPEND|os.O_CREATE|os.O_WRONLY, 0644)
    if err!=nil{
        log.Fatal(err)
    }
    defer file.Close()

    log.SetOutput(file)
    log.Println("Logging to file")
}
```

# STRUCTURED LOGGING

- Logs in key-value format (JSON)
- Easy for machines to read and analyze
- Example fields:
  - timestamp
  - level
  - message
  - userID

# POPULAR GO LOGGING LIBRARIES

- logrus – structured logging with levels
- zap – fast, production-ready logging
- zerolog – zero-allocation JSON logging
- slog – built-in structured logging in Go

# EXAMPLE WITH SLOG

```
package main

import "log/slog"

func main() {
    slog.Info("User logged in",
        "user", "admin",
        "role", "administrator",
    )
}
```

# BEST PRACTICES FOR LOGGING

- Use meaningful messages
- Include context (user, request ID, etc.)
- Avoid logging sensitive data (passwords, tokens)
- Use log levels correctly

# CHALLENGES IN LOGGING

- Too Many Logs (Noise)
  - Excessive logs make it hard to find important information
- Performance Overhead
  - Logging can slow down applications if done frequently
- Managing Large Log Files
  - Logs can grow very large over time making it hard to store, search and analyze manually
- Security and Privacy Concerns
  - Logs may contain sensitive data (passwords, tokens, personal data)

# CONCLUSION

- Logging is essential in Go applications
  - Helps developers understand how the program behaves
  - Makes it easier to detect and fix errors in production
- Go offers built-in and third-party logging tools
  - Standard log and slog packages for basic and structured logging
- Good logging improves debugging, monitoring, and security
  - Debugging: find bugs faster with detailed logs
  - Monitoring: track performance and system health
  - Security: detect suspicious activities and maintain audit trails

**THANK  
YOU**