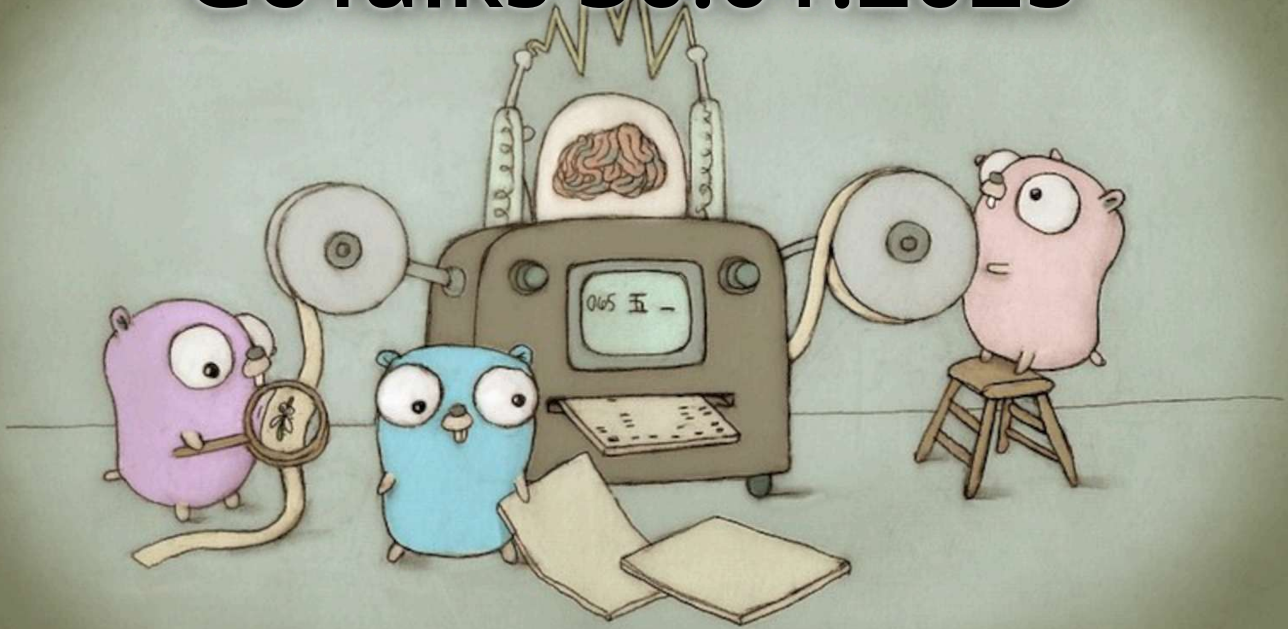
GoTalks 30.01.2025



How to reach us

meetup.com/Golang-ZG

@golangzg

github.com/golanghr/golangzg

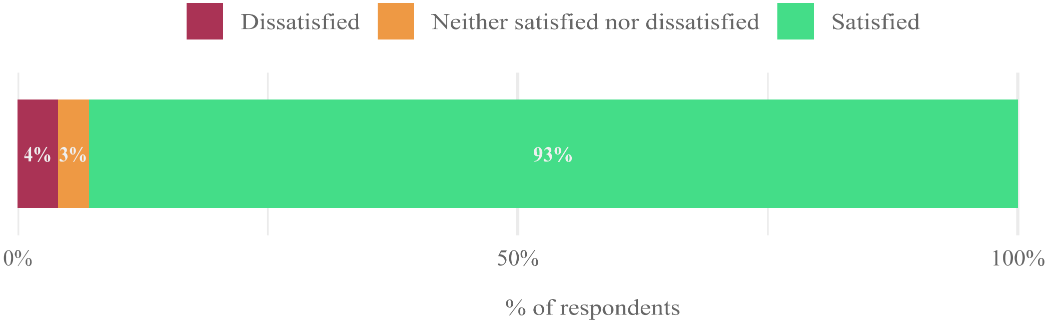
@golangzg.bsky.social

invite.slack.golangbridge.org

GolangZG

Go Developer Survey 2024 H2

Overall, how satisfied or dissatisfied have you felt while using Go during the past year?

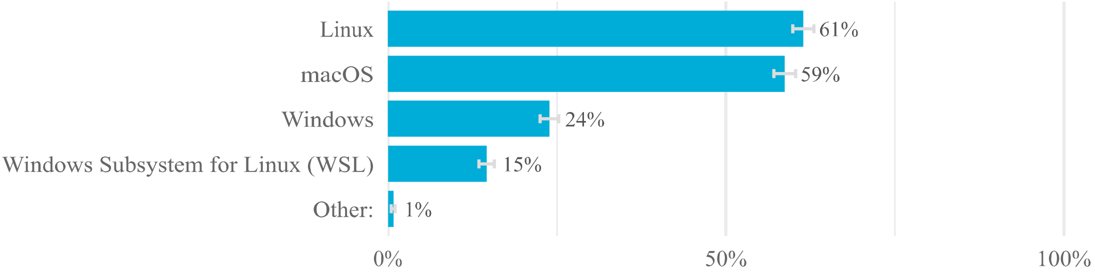


n = 3,705

Go Developer Survey 2024 H2

When writing Go code, I develop on:

(select all that apply)

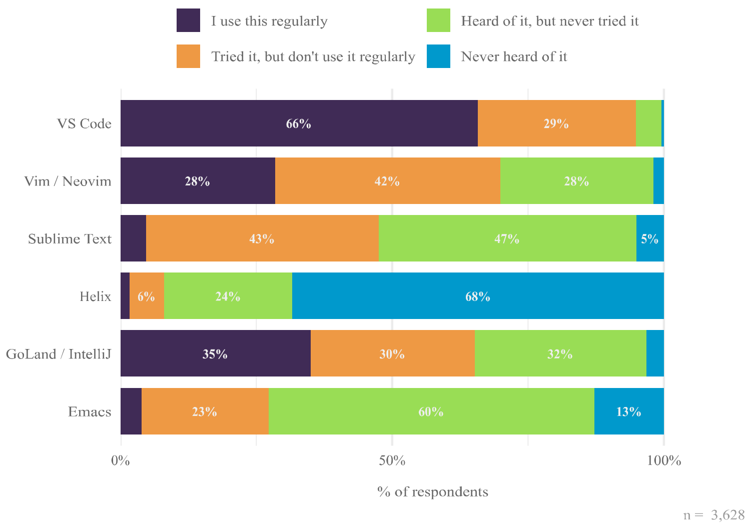


% of respondents

n = 3,676

GolangZG

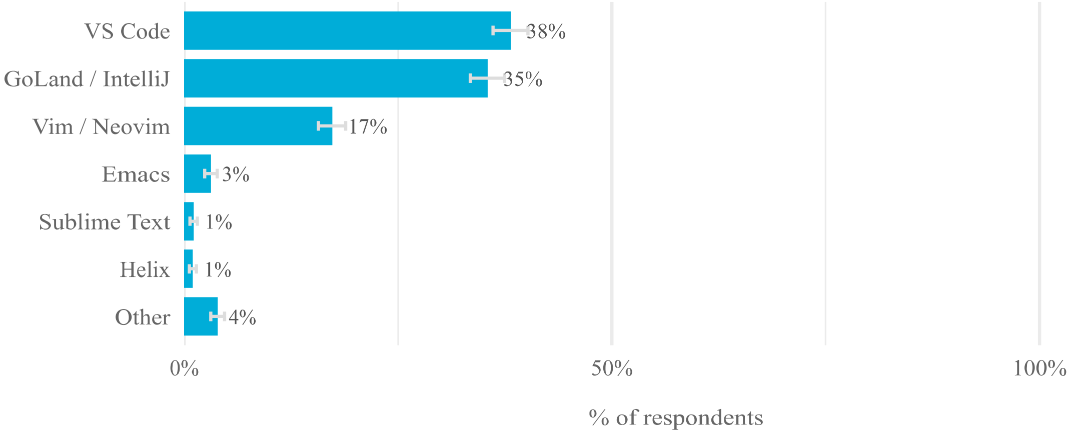
Go Developer Survey 2024 H2



How would you describe your familiarity with the following IDEs or code editors?

Go Developer Survey 2024 H2

My preferred editor for Go code is:



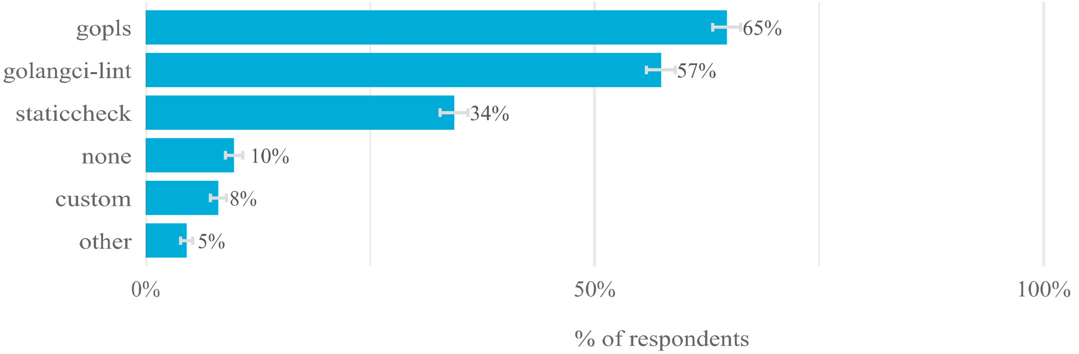
n = 2,18

GolangZG

Go Developer Survey 2024 H2

Which code analysis tools do you use on your Go code?

(select all that apply)

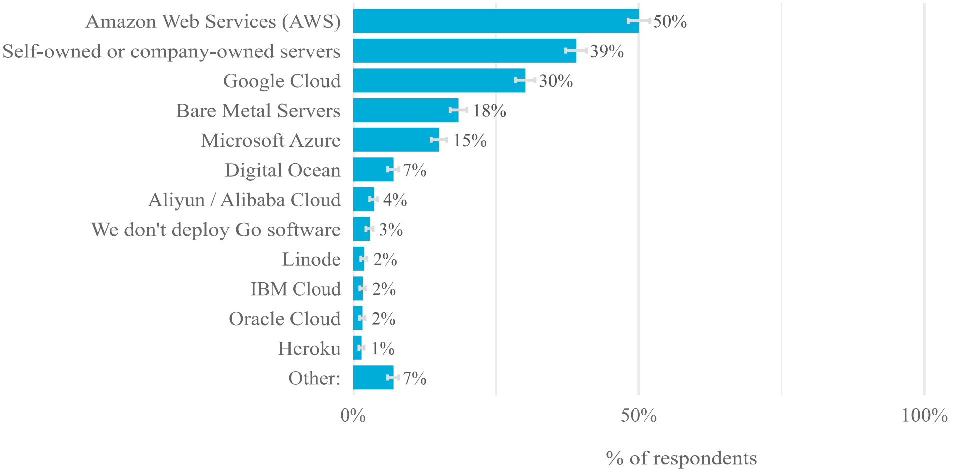


n = 3,64

Go Developer Survey 2024 H2

Does your team at work deploy Go programs to any of the following?

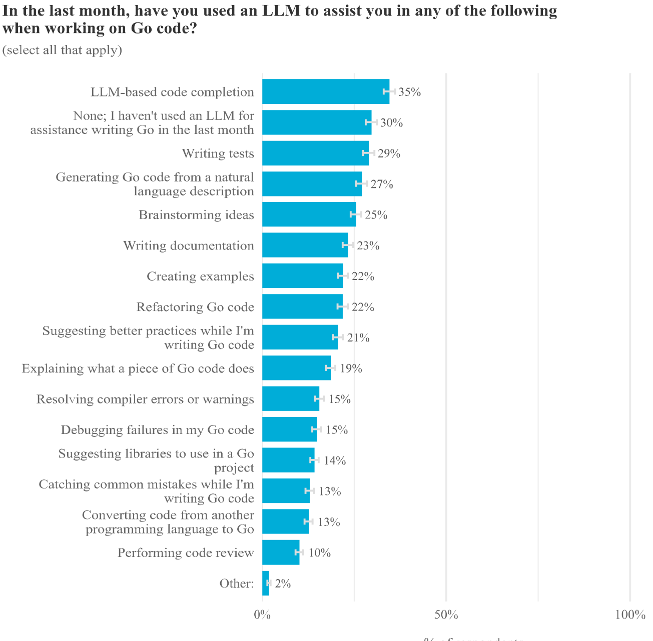
(select all that apply)



n = 2,723

GolangZG

Go Developer Survey 2024 H2

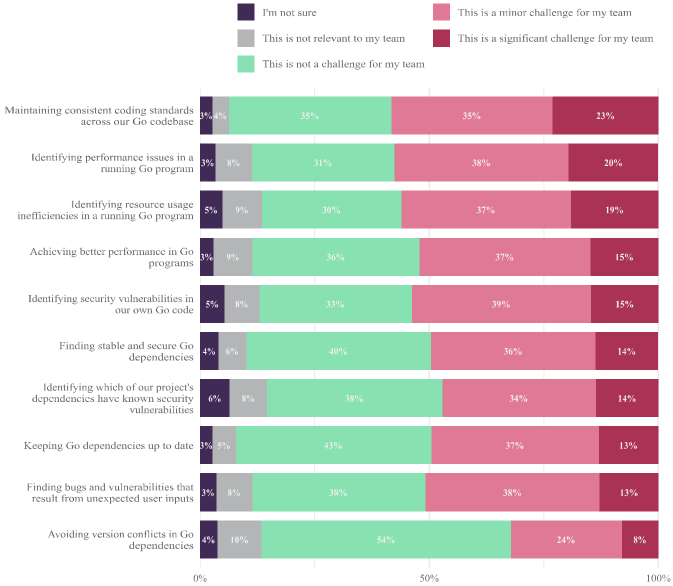


In the last month, have you used an LLM to assist you in any of the following when working on Go code?

(select all that apply)

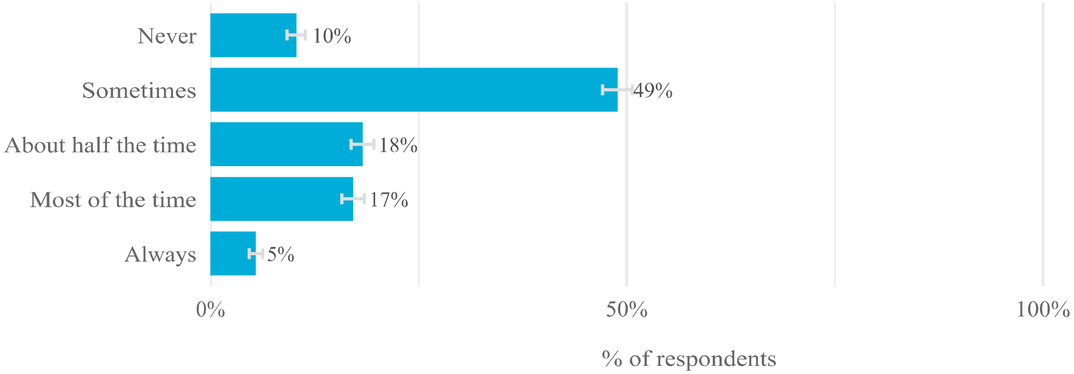
Go Developer Survey 2024 H2

At work, to what extent do the following challenges impact your team's current experience using Go?



Go Developer Survey 2024 H2

How often do you work on projects where performance optimizations are crucial?

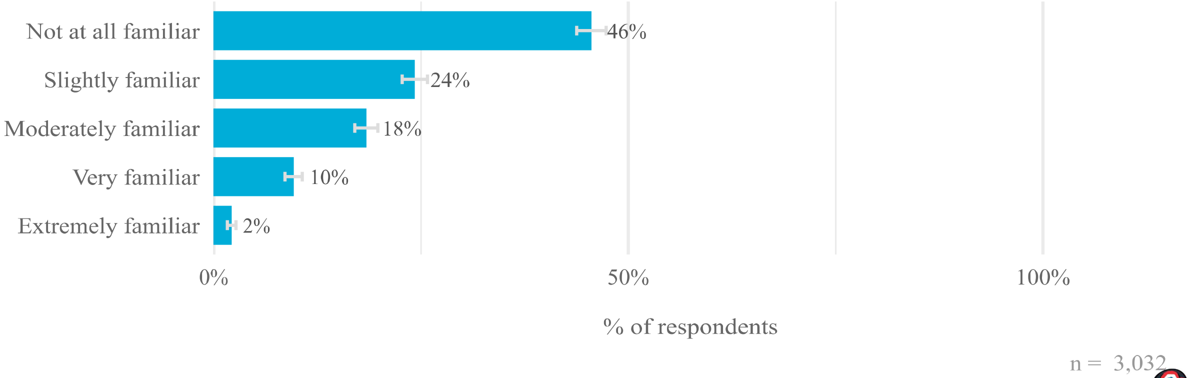


n = 3,025

GolangZG

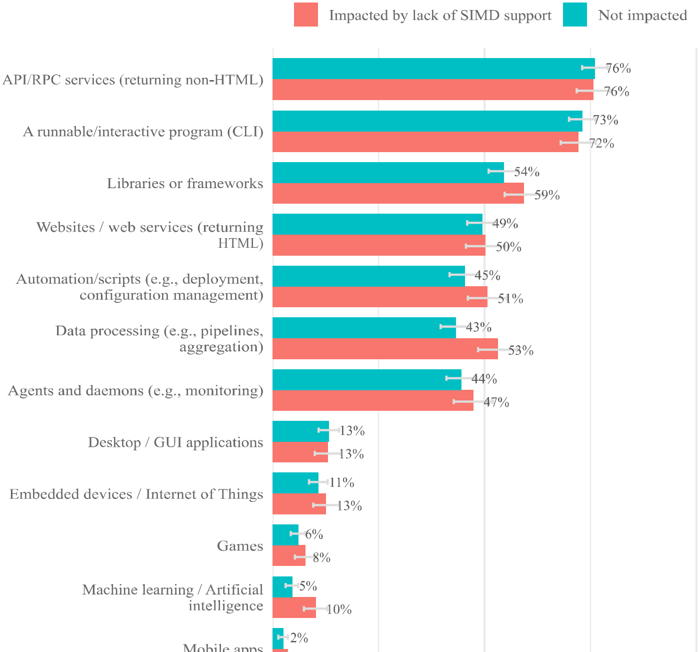
Go Developer Survey 2024 H2

How familiar are you with the concept of SIMD (Single Instruction, Multiple Data)?



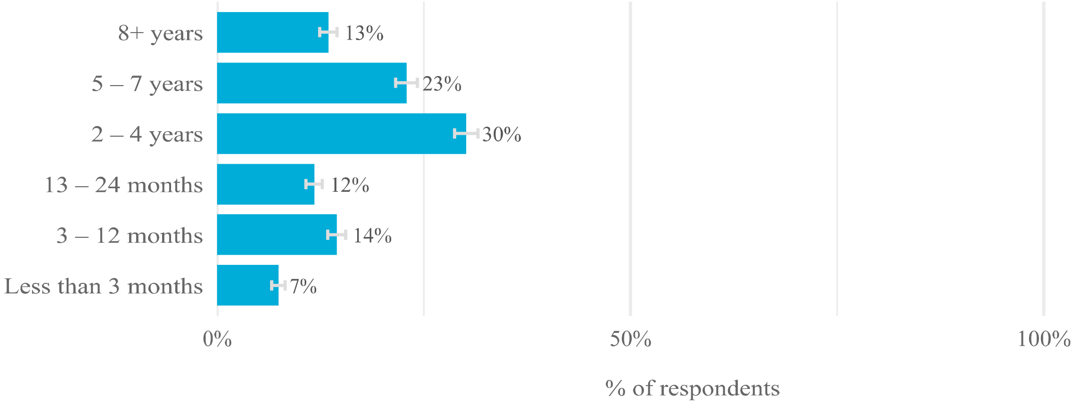
GolangZG

Go Developer Survey 2024 H2



Go Developer Survey 2024 H2

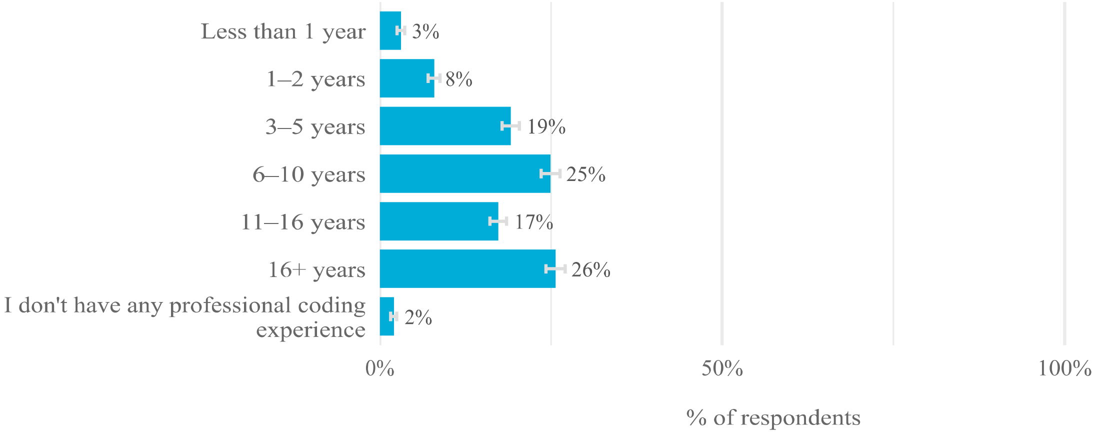
How long have you used Go?



n = 3,98

Go Developer Survey 2024 H2

How many years of professional coding experience do you have?



n = 3,7

Go Developer Survey 2024 H2

Highlights

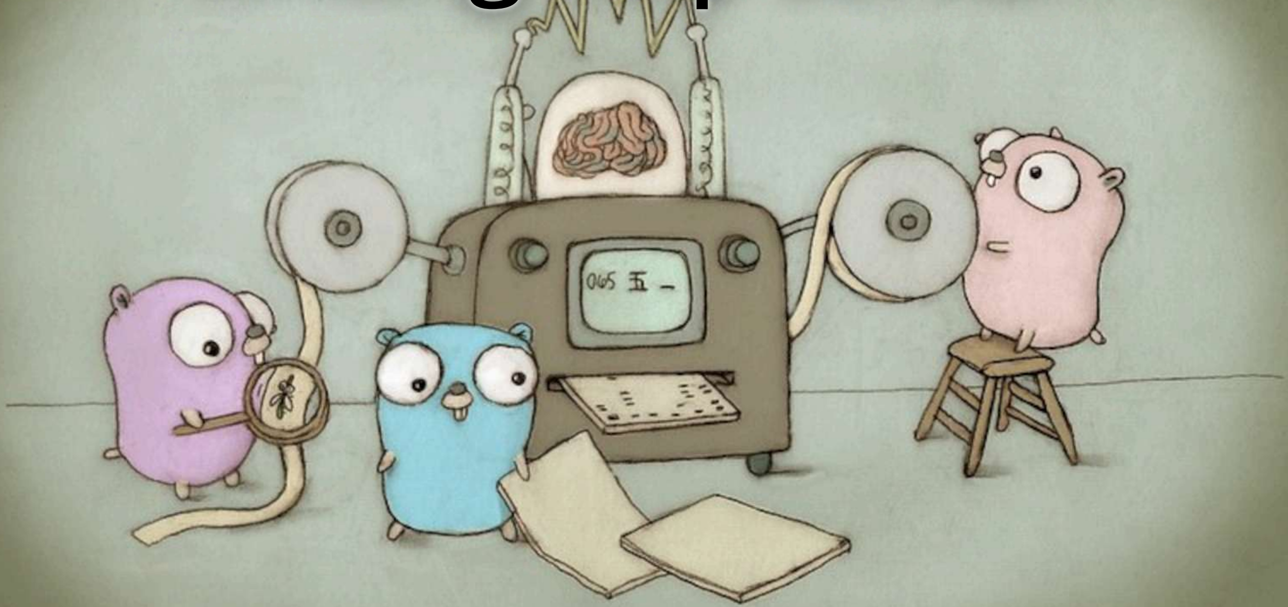
- 93% of survey respondents saying they felt satisfied while working with Go during the prior year.
- 70% of respondents were using AI assistants when developing with Go. The most common uses were:
  - LLM-based code completion
  - writing tests
  - generating Go code from natural language descriptions
  - brainstorming.

Go Developer Survey 2024 H2

Highlights

- Ease of deployment and an easy to use API/SDK for cloud providers.
  - First-class Go support is critical to keeping up with developer expectations.
- The biggest challenge for teams using Go was maintaining consistent coding standards across their codebase. This was often due to team members having different levels of Go experience and coming from different programming backgrounds, leading to inconsistencies in coding style and adoption of non-idiomatic patterns.

writing http server



web server - hello http

```go
import (
  "fmt"
  "net/http"
)

func helloWorld(w http.ResponseWriter, r *http.Request) {
  fmt.Fprint(w, "Hello World")
}

func main() {
  http.HandleFunc("/", helloWorld)
  http.ListenAndServe(":8080", nil)
}
```

GO web server - `hello http`

```go
func main() {
  http.Server{
    Addr:           ":8080",
    ReadTimeout:    5 * time.Second,
    WriteTimeout:   5 * time.Second,
    IdleTimeout:    5 * time.Second,
    MaxHeaderBytes: 1 << 20, // 1MB
    Handler:        http.HandlerFunc(helloWorld),
  }.ListenAndServe()
}
```

GO web server - `hello http`

```
server := &http.Server{
  Addr:         cfg.Address + ":" + strconv.Itoa(cfg.Port),
  ReadTimeout:  5 * time.Second,
  WriteTimeout: 5 * time.Second,
  IdleTimeout:  5 * time.Second,
}

err := server.ListenAndServe()
if err != nil && !errors.Is(err, http.ErrServerClosed) {
  log.Printf("HTTP server error: %s\n", err)
}
```

GO web server - graceful shutdown

```go
signalCh := make(chan os.Signal, 1)
signal.Notify(signalCh, syscall.SIGTERM, os.Interrupt)

go func() {
  log.Println("Listening on", server.Addr)
  err := server.ListenAndServe()
  // Handle the error
}()

<-signalCh
// Shutdown the server gracefully
log.Println("Shutting down...")
shutdownCtx, cancelShutdown := context.WithTimeout(...)
defer cancelShutdown()

server.SetKeepAlivesEnabled(false)
err := server.Shutdown(shutdownCtx)
// Handle the error
```

GO web server - static files

```
http.Handle("/{$}", Homepage(config))

//go:embed ui/index.html
var homePage []byte

func Homepage(config configuration.Config) http.Handler {
  return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    _, err := w.Write(homePage)
    if err != nil {
      log.Println(err)
      return
    }
  })
}
```

GO web server - `static files`

```go
//go:embed ui/static
var dist embed.FS

// Sub returns an [FS] corresponding to the subtree
// rooted at fsys's dir.
sub, err := fs.Sub(dist, "ui/static")
if err != nil {
  panic(err)
}
wd, err := os.Getwd()
if err != nil {
  panic(err)
}
handler := http.FileServer(http.FS(sub))
// ...
```

GolangZG

GO web server - static files - cache

```go
func Serve(w http.ResponseWriter, r *http.Request) {
  if r.Header.Get("If-None-Match") == eTag {
    w.WriteHeader(http.StatusNotModified)
    return
  }
  // ...
  rw.Header().Set("ETag", eTag)
  // ...
}
```

GolangZG

web server - content type

```go
func Serve(w http.ResponseWriter, r *http.Request) {
  // ...
  // Set the content type
  w.Header().Set("Content-Type", "text/html")
  // ...
}
```

GO web server - authentication

```
import (
  "golang.org/x/crypto/bcrypt"
)

func cookieAuth(password string, r *http.Request) (bool) {
  cookie, err := r.Cookie("present")
  if err != nil {
    return false
  }

  return bcrypt.CompareHashAndPassword(
      []byte(cookie.Value), []byte(password)) == nil
}
```

GolangZG

GO web server - middleware

```go
func AuthMiddleware(next http.Handler) http.Handler {
  return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    if !cookieAuth(config.Security.UserPwd, r) {
      http.Error(w, "Unauthorized", http.StatusUnauthorized)
      return
    }
    next.ServeHTTP(w, r)
  })
}
```

GO web server - middleware

```go
func AccessControlAllow(next http.Handler) http.Handler {
  return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    origin := r.Header.Get("Origin")
    if origin != "" {
      w.Header().Set("Access-Control-Allow-Origin", origin)
    }
    w.Header().Set("Access-Control-Allow-Methods",
        "OPTIONS, POST, GET, PUT, DELETE")
    w.Header().Set("Access-Control-Allow-Headers",
        "Accept, Content-Type, Content-Length, Authorization")
    next.ServeHTTP(w, r)
  })
}
```

web server - `middleware`

```go
func Recover(next http.Handler) http.Handler {
  return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    defer func() {
      if r := recover(); r != nil {
        log.Printf("recovered from panic: %v", r)
        http.Error(w, "Internal Server Error", http.StatusInternalServerError
      }
    }()
    next.ServeHTTP(w, r)
  })
}
```

web server - timeouts

```go
func SSE(server data.Server, config configuration.Config) http.Handler {
  return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    flusher, ok := w.(http.Flusher)
    if !ok {
      http.Error(w, "Streaming not supported!", http.StatusNotAcceptable)
      return
    }

    w.Header().Set("Content-Type", "text/event-stream")
    w.Header().Set("Cache-Control", "no-cache")
    w.Header().Set("Connection", "keep-alive")
    //  ...
    rc := http.NewResponseController(w)
    err = rc.SetReadDeadline(time.Now().Add(time.Minute))
    err = rc.SetWriteDeadline(time.Time{})
    // ...
    _, err := fmt.Fprintf(w, "%s\n\n", strings.Join(lines, "<br>"))
    // handle error ...
    flusher.Flush()
}
```

web server - certificates

```go
cert, err := tls.LoadX509KeyPair("server-cert.pem", "server-key.pem")
if err != nil {
  fmt.Println("Error loading certificate:", err)
  return
}

// Create an HTTPS server configuration
config := &tls.Config{Certificates: []tls.Certificate{cert}}
srv := &http.Server{
  Addr:      ":443",
  Handler:   http.HandlerFunc(helloHandler),
  TLSConfig: config,
}

err = srv.ListenAndServeTLS("", "")
// error handling
```

GolangZG

GO web server - certificates

```go
func getCertificates(*tls.ClientHelloInfo) (*tls.Certificate, error) {
  certMutex.RLock()
  defer certMutex.RUnlock()

  if len(certs) > 0 {
    return &certs[0], nil
  }
  return nil, nil
}

server := &http.Server{
  Addr: ":443",
  TLSConfig: &tls.Config{
    GetCertificate: getCertificates,
  },
  },
}
```

GolangZG