# Criando serviços em Golang

# Entendendo arquitetura de serviços

- Arquitetura de serviços é uma forma de estruturar sua aplicação tendo serviços que se comunicam através de APIs.
- Por característica facilita a escalabilidade, melhora a manutenção do código e permite implementação modular de novas funcionalidades.
- Serviços são componentes individuais que performam alguma task específica e interagem com outros por APIs.
- Diferente de arquiteturas monolíticas, onde tudo é acoplado, é promovido a diminuição do acoplamento e separação de responsabilidades.

# Por que Golang para Arquitetura de Serviços?

- Why Golang? Concorrência, performance, simplicidade e um forte ecossistema o torna ideal para essa abordagem.
- Key Components: Goroutines para concorrência, channels para comunicação, packages para modularidade e interfaces para flexibilidade.
- Ferramentas populares como Docker, Kubernetes e gRPC aprimoram ainda mais os recursos da Golang no desenvolvimento de serviços.

# Routing e Handling HTTP Requests em Golang

- Routing: Direciona requisições HTTP recebidas para os seus específicos handlers.
- net/http: built-in package do Golang para lidar com routing e handling de requests HTTP.
- Request Handling: Processa uma request e manda uma resposta de volta.

```
    // Cria um logger com saída JSON
    logger := slog.New(slog.NewJSONHandler(os.Stdout, nil))

    // Cria um roteador HTTP
    mux := http.NewServeMux()

    // Rotas POST com autenticação básica
    mux.HandleFunc("/cash-out", basicAuth(handleRoute(logger, "pix-cash-out")))
    mux.HandleFunc("/cash-in", basicAuth(handleRoute(logger, "pix-cash-in")))
    mux.HandleFunc("/reversal-out", basicAuth(handleRoute(logger, "pix-reversa
    mux.HandleFunc("/reversal-in", basicAuth(handleRoute(logger, "pix-reversal

    port := os.Getenv("PORT")

    if port == "" {
        port = "8080"
    }

    // Inicia o servidor
    logger.Info("Servidor iniciado", "port", port)
    _ = http.ListenAndServe(fmt.Sprintf(":%s", port), mux)
}

// Middleware de autenticação básica
func basicAuth(next http.HandlerFunc) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        username, password, ok := r.BasicAuth()
```

Exemplo 1

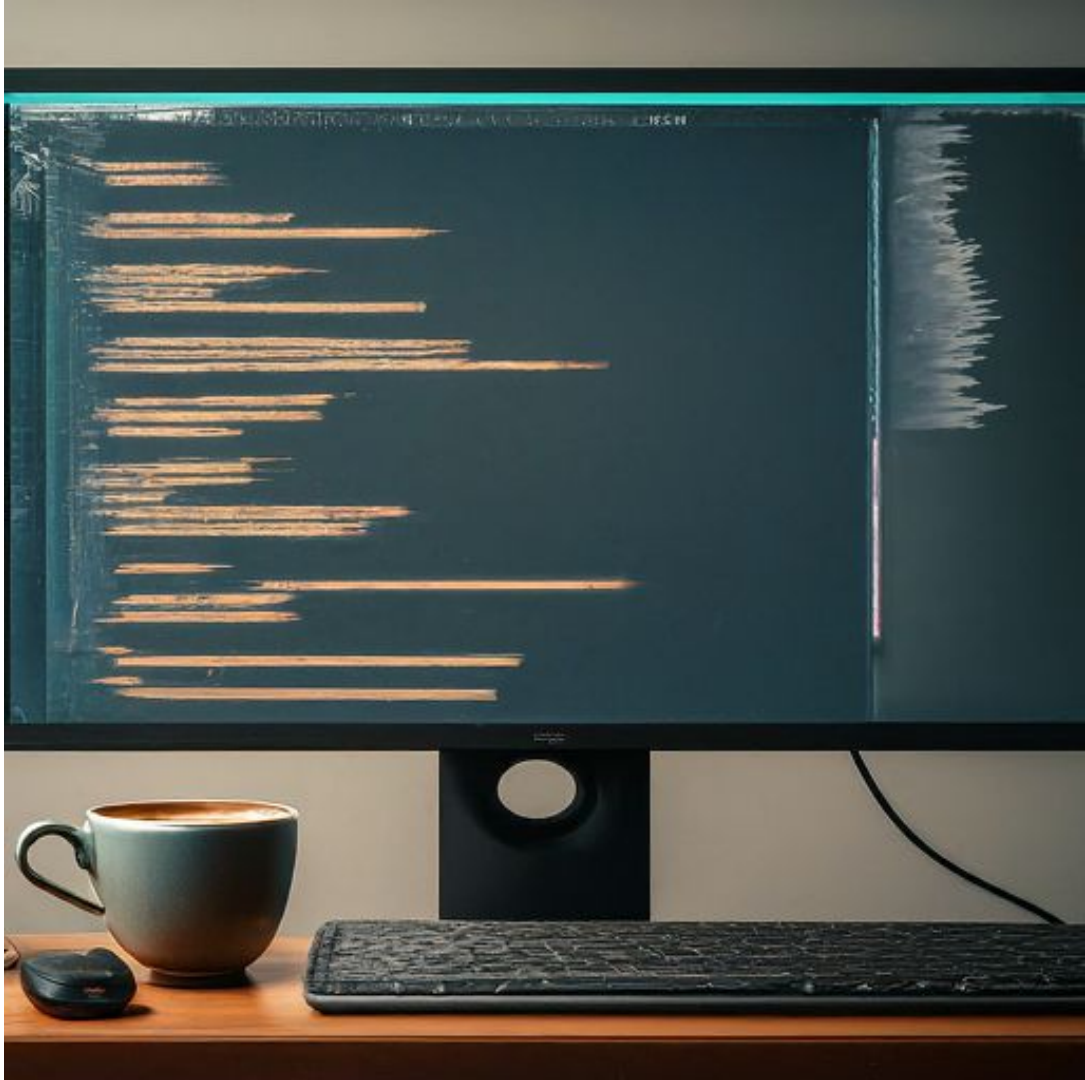
# Serialization and Deserialization of Data (JSON) in Golang

- JSON: lightweight data-interchange format, common in APIs and web dev.
- Serialization: Golang data structures to JSON format.
- Key serialization functions: json.Marshal(), json.MarshalIndent().
- Deserialization: JSON data to Golang data structures.
- Key deserialization function: json.Unmarshal().

Exemplo 2 e 3

# Logs estruturados

- Logs estruturados são um formato padronizado de log que utiliza pares chave-valor.
- Benefícios: Melhor capacidade de busca, facilidade de depuração e melhor integração com ferramentas de análise de logs.
- log/slog é uma biblioteca nativa para logs estruturados
- Logging contextual: permite adicionar informações contextuais aos logs.
- Níveis de log personalizáveis: suporte a vários níveis de log configuráveis.
- Suporte para vários formatos de saída: flexibilidade na forma como os logs são exibidos e armazenados.



Exemplo 4