📘 Assignment 4 – Golang Design Patterns, Architecture, and Best Practices

Welcome to Assignment 4 of GoFr Summer of Code! This assignment dives into some core software engineering concepts using Golang. By the end of this, you'll be exposed to real-world architectural principles that power scalable services.

🔹 Reading Material & Concepts to Cover

1. Design Patterns in Golang

Explore the following design patterns that are extremely useful in microservices:

Factory Pattern

- Understanding the Factory Method Pattern in Go: Understanding the Factory Method Pattern in Go - DEV Community
- Factory Method - Refactoring.Guru: Factory Method

Adapter Pattern

- Adapter in Go - Refactoring.Guru: Adapter in Go / Design Patterns

Optional Pattern in Go

- Golang 101: Option Pattern: Golang 101: Option Pattern

Chain of Responsibility Pattern

- Chain of Responsibility Pattern in Go: Elevate Code with Chain of Responsibility pattern in Go: A Master Guide in 2024

Strategy Pattern

- Understanding the Strategy Pattern in Go: Understanding the Strategy Pattern in Go — Coding Explorations

Singleton Pattern

- Singleton Design Pattern: Singleton Design Pattern: Creating Unique Instances Efficiently

Builder Pattern

- Builder in Go - Refactoring.Guru: Builder in Go / Design Patterns
- Understanding the Builder Pattern in Go: Understanding the Builder Pattern in Go - DEV Community

Decorator Pattern

- Golang Decorator Pattern: Golang Decorator Pattern | Henry Du Blog
- How Decorator Pattern is used in Building GoFr | by Aryan Mehrotra | Stackademic

2. Interfaces in Golang

Interfaces provide powerful abstractions in Go. Here's what to cover:

- When and why to use interfaces in Golang
- When not to use interfaces (e.g., for types that won't vary)
- Use cases where interfaces enhance modularity, testability, and mocking

📖 Read:

- How To Use Interfaces in Go | DigitalOcean
- Golang Interfaces Explained – Alex Edwards
- Understanding the Power of Go Interfaces: A Comprehensive Guide | by Jamal Kaksouri | Medium
- Why Use GoFr for Golang Backend? | by Aryan Mehrotra | Level Up Coding
- Ben Johnson's Interface Usage Discussion: Ben Johnson's WTF project layout: interface usage : r/golang

3. Three Layer Architecture

Understand how three-layer architecture uses the following structure:

- Handler: Request-response layer (Presentation Tier)
- Service: Business logic (Application Tier)
- Store: Database/persistence logic (Data Tier)

📖 Docs:

- What Is Three-Tier Architecture? : Clean Architecture in Go: Clean Architecture in Go

- Building RESTful APIs in Golang: The Microservices Backlash: Over-Engineering or Misunderstood Architecture? | by Aryan Mehrotra | Apr, 2025 | Stackademic

4. SOLID Principles in Golang

Understand and apply the five SOLID principles:

- S: Single Responsibility Principle
- O: Open-Closed Principle
- L: Liskov Substitution Principle
- I: Interface Segregation Principle
- D: Dependency Inversion Principle

📖 Read:

- SOLID Principles in Go with Code Examples: GO Design Patterns: An Introduction to SOLID | HackerNoon
- SOLID Go Design | Dave Cheney
- Reddit discussion SOLID principles in Go with code examples : r/golang

5. Dependency Injection in Go

Get a feel for how DI can help decouple components and improve testability.
📖 Read:

- Dependency Injection in Go - Part 1 - JetBrains Guide

🧠 Optional Assignment:

Take some time to explore **Dependency Injection (DI)** in Golang — we'll be diving deep into it in the next assignment, especially while implementing **Test-Driven Development (TDD)**.

As a reminder: **writing tests is mandatory** for any code contributed to the GoFr repository. So it's time to build the habit early. Trust us, your future self (and your reviewers) will thank you!

Start reading, get curious — it'll all come together soon. 👨‍💻🧪

🧪 Assignment Task (Hands-on)

You need to implement a simple blog post service using the above concepts.

Problem Statement:

Build a Blog Service that supports:

- Create blog post
- Get all posts
- Get a post by ID

Requirements:

- Use three-layer architecture (Handler, Service, Store)
- Use at least one design pattern (Factory, Adapter, Optional, Chain, etc.)
- Apply at least 3 SOLID principles in your design
- Inject dependencies (e.g., service into handler, store into service)
- Demonstrate at least one useful interface-based abstraction
- Now that you already know how to spin and connect to MySQL and Redis use the same in your database layer.

🔁 Deliverables:

- Code pushed to a GitHub repository
- README with architecture explanation
- Mention which design pattern, SOLID principles, and interface use cases were applied

Happy Coding!
— GoFr SoC Team 🚀