# Designing a Production-Ready REST API in Go (Golang): Clean Architecture, PostgreSQL, and Concurrency in Action

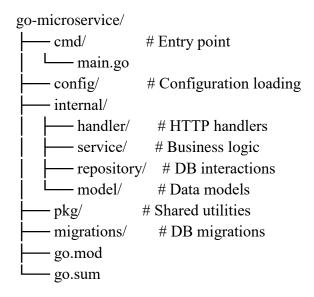
### Introduction

Using PostgreSQL for tenacity, clean architecture principles, and Go's built-in concurrent features, we will create a ready for production RESTful API in this extensive tutorial. This tutorial is not just about coding — it's about understanding the architecture, performance, and maintainability that tech companies expect.

# **Project Overview**

We'll create a task management API that allows CRUD operations on tasks, supports concurrent processing for background jobs, and includes middleware for logging and request timeouts.

# **Folder Structure (Clean Architecture)**



# Step 1: go.mod Initialization

go mod init github.com/maria/go-microservice

```
go get github.com/gorilla/mux
go get github.com/jmoiron/sqlx
go get github.com/lib/pq
go get github.com/joho/godotenv
```

### Step 2: Database Model (internal/model/task.go)

# Step 3: Repository Layer (internal/repository/task\_repository.go)

```
import (
          "github.com/jmoiron/sqlx"
          "your_module/internal/model"
)

type TaskRepository interface {
          Create(task model.Task) error
          GetAll() ([]model.Task, error)
          Update(task model.Task) error
          Delete(id int) error
}

type taskRepo struct {
          db *sqlx.DB
}
```

```
func NewTaskRepository(db *sqlx.DB) TaskRepository {
    return &taskRepo{db: db}
}
// Implementation of repository methods...
```