# Golang Gin Gorm Starter

Contribute to Caknoo <https://github.com/Caknoooo/go-gin-clean-starter>

## Introduction 👋

Clean Architecture is an approach to organizing code in an application that focuses on separating responsibilities and dependencies between components. In the context of Golang, Clean Architecture refers to the application of Clean Architecture principles in developing applications using the Go programming language.

Clean Architecture proposes a structured application design with several layers that have clear and isolated responsibilities. Each layer has a specific role and boundaries. Here are some common layers in Golang Clean Architecture:

## Directory / Layers 🔥

* **Config**: Aims to be directly related to things outside the code, such as databases. Configuration files play a crucial role in customizing the behavior of software applications. A well-structured config file can simplify the process of fine-tuning various settings to meet specific project requirements.
* **Constants**: This directory deals with things that cannot be changed, in other words, it is always constant and is usually called repeatedly.
* **Middleware**: An intermediary layer that serves to process and modify HTTP requests as they pass through the server before reaching the actual routes or actions. Middleware can be used to perform various tasks such as user authentication, data validation, logging, session management, response compression, and many more. It helps separate different functionalities within the API application and enables consistent processing for each incoming HTTP request.
* **Controller**: A component or part of the application responsible for managing incoming HTTP requests from clients (such as browsers or mobile applications). The controller controls the flow of data between the client and the server and determines the actions to be taken based on the received requests. In other words, a controller is a crucial part of the REST API architecture that governs the interaction between the client and the server, ensuring that client requests are processed correctly according to predefined business rules.
* **Service**: A component responsible for executing specific business logic or operations requested by clients through HTTP requests. The service acts as an intermediary layer between the controller and data storage, fetching data from storage or performing the relevant business operations, and then returning the results to the controller to be sent as an HTTP response to the client. The significance of service in REST API architecture is to separate the business logic from the controller, making the application more modular, testable, and adaptable. In other words, services enable the separation of responsibilities between receiving HTTP requests (by the controller) and executing the corresponding business actions. This helps maintain clean and structured code in the development of RESTful applications.
* **Repository**: A component or layer responsible for interacting with data storage, such as a database or file storage, to retrieve, store, or manage data. The repository serves as a bridge between the service and the actual data storage. The primary function of a repository is to abstract database or storage-related operations from business logic and HTTP request handling. In other words, the repository provides an interface for accessing and manipulating data, allowing services to focus on business logic without needing to know the technical details of data storage underneath. In the architecture of a REST API, the use of repositories helps maintain separation of concerns between different tasks in the application, making development, testing, and code maintenance more manageable.
* **Utils**: Short for "utility functions" or "utility tools," this refers to a collection of functions or tools used for common tasks such as data validation, string manipulation, security, error handling, database connection management, and more. Utils help avoid code duplication, improve code readability, and make application development more efficient by providing commonly used and reusable functions.

## Prerequisite 🏆

* Go Version >= go 1.20
* MySQL Version >= version 8.0

## How To Use

There are 2 ways to do running

### With Docker

1. Copy the example environment file and configure it:

cp.env.example .env

1. Build Docker

docker-compose build --no-cache

1. Run Docker Compose

docker compose up -d

### Without Docker

1. Clone the repository or **Use This Template**

git clone https://github.com/Software-House-MCI-ITS/TEMPLATE-BE

1. Navigate to the project directory:

cd TEMPLATE-BE

1. Copy the example environment file and configure it:

cp .env.example .env

1. Configure .env with your PostgreSQL credentials:

DB\_HOST=localhost

DB\_USER=mysql

DB\_PASS=

DB\_NAME=gin\_example\_db

DB\_PORT=5432

1. Run the application:

go run main.go

## Run Migrations and Seeder

To run migrations and seed the database, use the following commands:

go run main.go --migrate --seed

#### Migrate Database

To migrate the database schema

go run main.go --migrate

This command will apply all pending migrations to your MySQL database specified in .env

#### Seeder Database

To seed the database with initial data:

go run main.go --seed

This command will populate the database with initial data using the seeders defined in your application.

## What did you get?

By using this template, you get a ready-to-go architecture with pre-configured endpoints. The template provides a structured foundation for building your application using Golang with Clean Architecture principles.

### Postman Documentation

You can explore the available endpoints and their usage in the [Postman Documentation](https://documenter.getpostman.com/view/4071385/2sAXjSz8xB). This documentation provides a comprehensive overview of the API endpoints, including request and response examples, making it easier to understand how to interact with the API.

### Issue / Pull Request Template

The repository includes templates for issues and pull requests to standardize contributions and improve the quality of discussions and code reviews.

* **Issue Template**: Helps in reporting bugs or suggesting features by providing a structured format to capture all necessary information.
* **Pull Request Template**: Guides contributors to provide a clear description of changes, related issues, and testing steps, ensuring smooth and efficient code reviews.