


# Clean Architecture using Golang

 [medium.com/@eminetto/clean-architecture-using-golang-b63587aa5e3f](https://medium.com/@eminetto/clean-architecture-using-golang-b63587aa5e3f)

## What is Clean Architecture?

In his book *"Clean Architecture: A Craftsman's Guide to Software Structure and Design"* famous author Robert "Uncle Bob" Martin presents an architecture with some important points like testability and independence of frameworks, databases and interfaces.

The constraints in the Clean Architecture are :

- Independent of Frameworks. The architecture does not depend on the existence of some library of feature laden software. This allows you to use such frameworks as tools, rather than having to cram your system into their limited constraints.
- Testable. The business rules can be tested without the UI, Database, Web Server, or any other external element.
- Independent of UI. The UI can change easily, without changing the rest of the system. A Web UI could be replaced with a console UI, for example, without changing the business rules.
- Independent of Database. You can swap out Oracle or SQL Server, for Mongo, BigTable, CouchDB, or something else. Your business rules are not bound to the database.
- Independent of any external agency. In fact your business rules simply don't know anything at all about the outside world.

More at <https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html>

So, based on this constraints, every layer must be independent and testable.

From Uncle Bob's Architecture we can divide our code in 4 layers :

- Entities: encapsulate enterprise wide business rules. An entity in Go is a set of data structures and functions.
- Use Cases: the software in this layer contains application specific business rules. It encapsulates and implements all of the use cases

of the system.

- **Controller:** the software in this layer is a set of adapters that convert data from the format most convenient for the use cases and entities, to the format most convenient for some external agency such as the Database or the Web
- **Framework & Driver:** this layer is generally composed of frameworks and tools such as the Database, the Web Framework, etc.

## Clean Architecture in Golang

Let's use as an example the package *user*:

In the file *entity.go* we have our entities:

In the file *repository.go* we have the interface that defines a repository, where the entities will be stored. In this case the repository means the *Framework & Driver* layer in Uncle Bob architecture. Its content is:

This interface can be implemented in any kind of storage layer, like MongoDB, MySQL, and so on. In our case we implemented using MongoDB, as seen in *mongodb.go*:

The file *service.go* represents the *Use Case* layer, as defined by Uncle Bob. In the file we have the interface *Service* and its implementation. The *Service* interface is:

The last layer, the *Controller* in our architecture is implemented in the content of *api*:

In the following code, from *api/main.go*, we can see how to use the services:

Now we can easily create tests to our packages, like:

Using the Clean Architecture we can change the database from MongoDB to Neo4j, for instance, without breaking the rest of application. And we can grow our software without losing quality and speed.

## References

<https://hackernoon.com/golang-clean-architecture-efd6d7c43047>

<https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html>