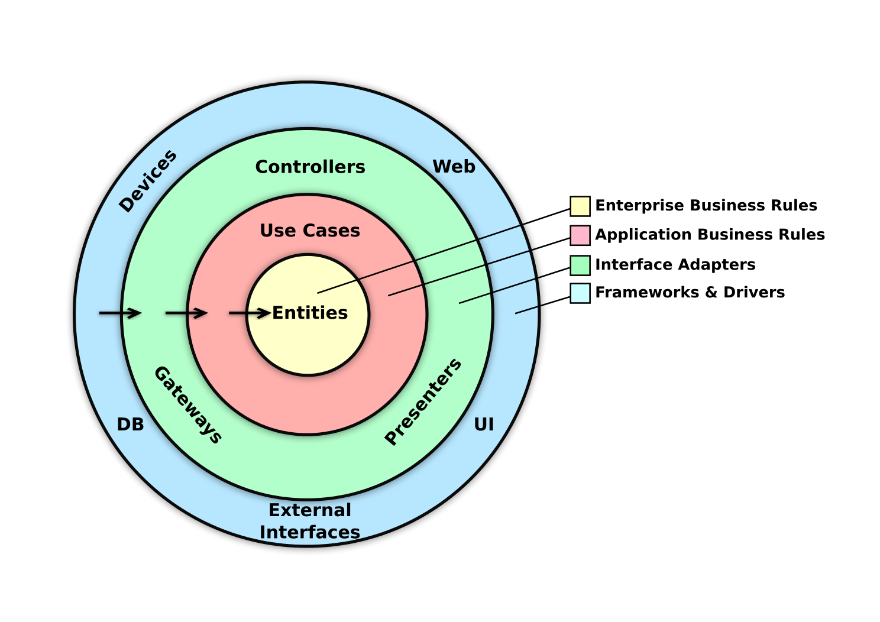
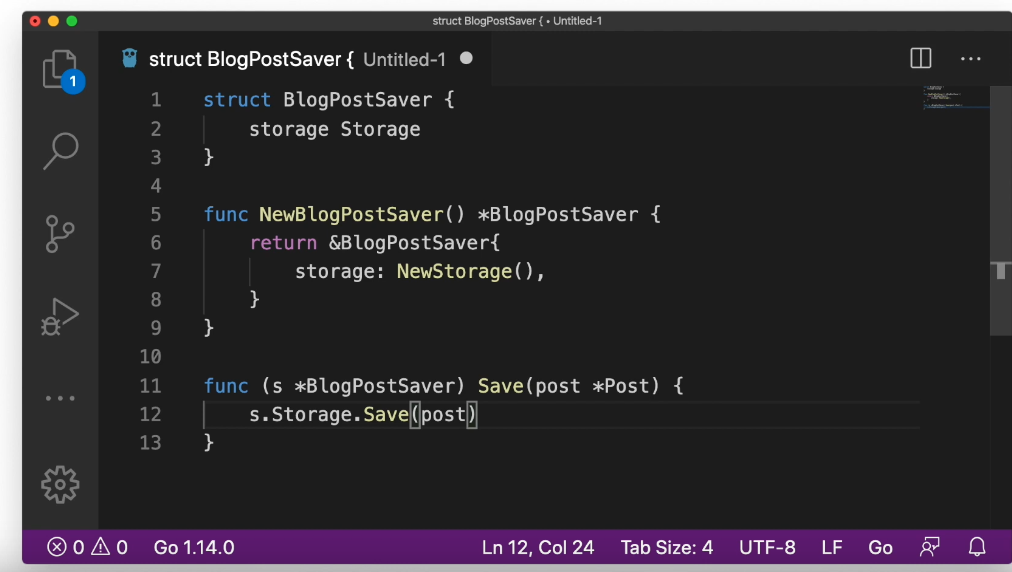
# Чистая архитектура:

Смысл чистой архитектуры в том, что в проекте выстроена иерархия слоев, где каждый слой отвечает за свои обязанности.

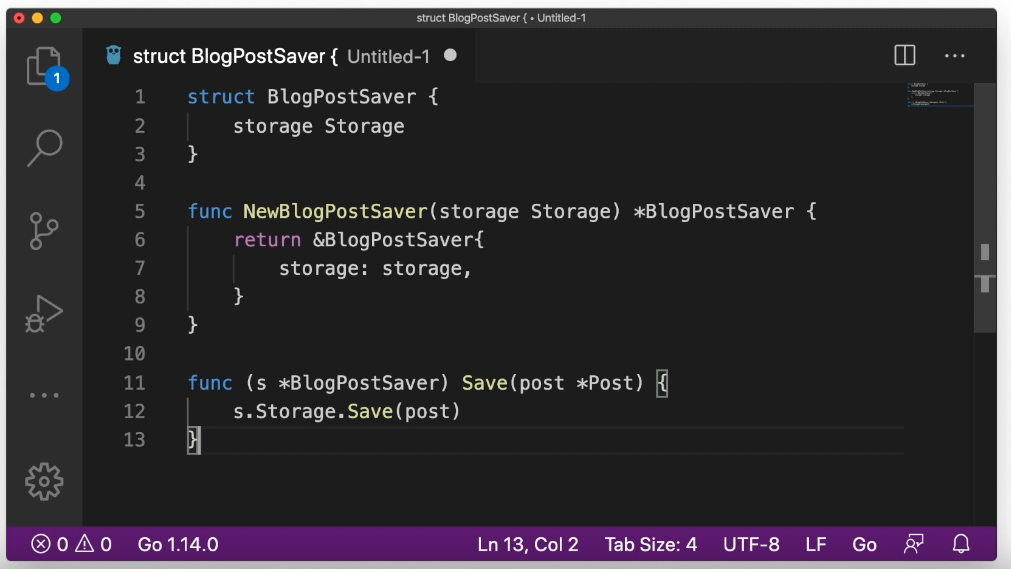
Зависимости слоев выстроены так, что внешний слой зависит от внутреннего, но внутренний не зависит от внешнего. Пример:



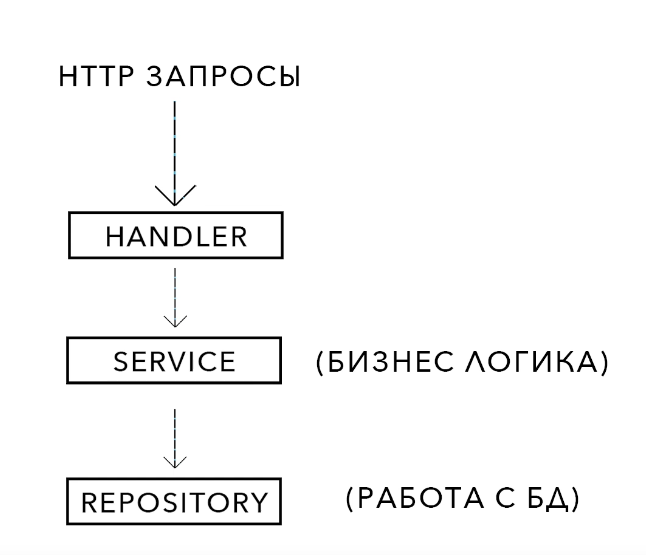
Пример в Golang:

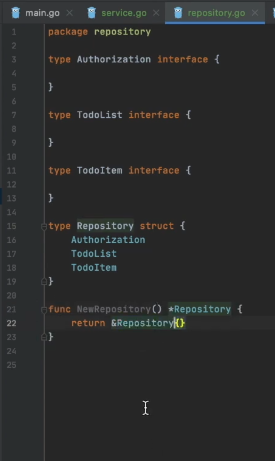


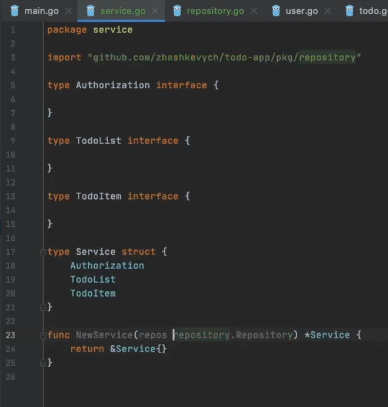
При создании структуры мы не создаем нужный нам объект, от которого зависит слой внутри конструктора. Вместо этого мы передаем в конструктор интерфейс, которому в дальнейшем будет удовлетворять объект, который мы будем использовать. Это есть хороший пример метода Dependency Injection – внедрения зависимостей.



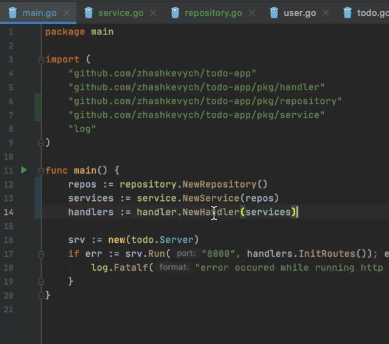
Причина использования интерфейсов заключается в том, что мы можем передавать не объект определенного класса, а объект любого класса, который удовлетворяет интерфейсу.







Объект сервиса зависит от объекта репозитория, поэтому мы передаем в его конструктор объект Repository



12-14 строки – создаем Repository -> создаем Service, который зависит от Repository -> создаем Handler, который зависит от Service.

# Файлы конфигураций. Работа с БД в Docker. Файлы миграций.:

**Файлы конфигураций** нужны для того, чтобы не указывать определенные данные в самом коде приложения. Это полезно, т.к. мы можем использовать разные конфигурации в зависимости от текущей ситуации (например разные окружения).

Для создания и прописывания конфигураций в Golang нужно создать папку configs, а в ней файл config.yml. Для работы с конфигурациями можно использовать библиотеку Viper. (go get –u github.com/spf13/viper)

**Для работы с базой данных можно использовать Docker, запустив БД в отдельном контейнере.**

**Для подключения к контейнеру используется exec –it.**

**Работа с миграциями.** Миграции являются своего рода системой контроля версий базы данных. При создании миграций создается 2 файла – up и down. Up-файл нужен для добавления изменений в БД. Down-файл используется для отката изменений, созданных при помощи up-файла.

**При написании миграций важно учитывать очередность выполнения команд.**

Для работы с миграциями в Golang можно использовать утилиту golang-migrate.