

**Arquitectura de Software I**

**GUÍA de TRABAJOS PRÁCTICOS**

**2025**

**Agustina Edmé Aliciardi**

**Eduardo Gaite**

**Emiliano Kohmann**

**Gerardo Salinas**

Trabajos Prácticos

# Índice

[TRABAJO PRÁCTICO Nº 1](#_1fob9te)[TRABAJO PRÁCTICO Nº 2](#_4d34og8)[TRABAJO PRÁCTICO Nº 3](#_35nkun2)[TRABAJO PRÁCTICO Nº 4](#_1y810tw)[TRABAJO PRÁCTICO Nº 5](#_3as4poj)[TRABAJO PRÁCTICO Nº 6](#_3o7alnk) 22

# 

**ENCUADRAMIENTO DE LA FORMACIÓN PRÁCTICA**

La formación práctica que se desarrolla en esta GTP incluye la resolución de problemas tipo o rutinarios de problemas de Ingeniería.

**2. CONSIGNAS GENERALES**

Los problemas propuestos en el presente TP deben ser resueltos en forma individual, y el código debe ser subido a github de manera pública.

De acuerdo a la complejidad creciente que esta asignatura presenta, el planteo o formulación de los problemas debe ser realizado basándose en lo desarrollado en las clases teóricas referidas a la unidad correspondiente y a todas las unidades anteriores ya desarrolladas, como también de aquellas asignaturas que son correlativas de esta.

Es fundamental para un correcto aprendizaje que el alumno controle sus soluciones con la herramienta de desarrollo elegida ([VisualStudio](https://code.visualstudio.com/), [Postman](https://www.postman.com/), [MySQL](https://www.mysql.com/downloads/), etc).

**3. PRESENTACIÓN DE LOS TRABAJOS PRÁCTICOS**

**3.1 Ejercicios de clase**

Todos los ejercicios realizados en clase emplearán código base disponible en la plataforma de github: <https://github.com/UCC-ArqSoft1/ejercicios2025>

**3.2 Proyecto**

De manera evolutiva, es decir, clase a clase se irá realizando un proyecto con tecnologías web: Go, React y MySQL. El mismo debe ser presentado en la plataforma de GitHub [https://github.com/UCC-ArqSoft1/proyecto2025-[Alumno1]-[Alumno2]](https://classroom.github.com/a/imdgYQQs)

El trabajo debe ser realizado de manera individual, ó en grupo de hasta 2 personas. El primer parcial consistirá en código go del backend (se evaluará igualmente el buen uso de git).

El segundo parcial consistirá en el frontend desarrollado en react, además de las correcciones y observaciones realizadas durante el primer parcial.

UNIDAD I: Introducción a las aplicaciones web

Son herramientas clave, ya que permiten a los usuarios acceder a servicios y funcionalidades a través de Internet, sin necesidad de instalar software adicional.

Estas aplicaciones están desarrolladas en lenguajes interpretables por los navegadores web y funcionan dentro de un esquema compuesto por tres elementos fundamentales: el cliente, que ejecuta la interfaz en el navegador; el servidor, que procesa la lógica de la aplicación; y la base de datos, donde se almacena y gestiona la información.

Gracias a esta arquitectura, las aplicaciones web ofrecen soluciones flexibles, escalables y de fácil acceso, siendo esenciales en el desarrollo de servicios digitales modernos.

# TRABAJO PRÁCTICO Nº 1 Análisis de Servicios

**Objetivos de Aprendizaje**

* Conocer la estructura de las peticiones y respuestas.
* Adquirir habilidad en la interpretación de la información que poseen las peticiones y respuestas.

**Unidad temática que incluye este trabajo práctico**

Este trabajo práctico corresponde a la unidad Nº 1 de la programación de la asignatura.

**Consignas a desarrollar en el trabajo práctico**

**Sugerencia:** muchos libros están disponibles digitalmente a través de la plataforma e-libro. Consulta en biblioteca.

**Preguntas sobre Aplicaciones Web:**

* 1. ¿Qué son las aplicaciones web?
  2. ¿Qué capas posee la arquitectura cliente-servidor? Explique cada una.
  3. ¿Qué es el frontend y el backend? ¿Qué tecnologías/frameworks se pueden emplear para su desarrollo?
  4. ¿Qué es el protocolo HTTP? ¿Cómo se componen las peticiones y respuestas? ¿Qué códigos de respuestas puede responder un servicio y con qué tipos de mensajes está relacionado?
  5. ¿Qué significa REST? ¿Qué operaciones se pueden realizar?
  6. ¿Qué significa API? Nombre sus ventajas.
  7. ¿Qué es Swagger? ¿Para qué sirve?
  8. ¿Qué es Postman? ¿Para qué sirve?
  9. ¿Qué significa cURL?
  10. ¿Qué es el modelo MVC?



Para resolver los ejercicios debes contar con: [Postman](https://www.postman.com/)

## Ejercicios de Servicios:C:\Users\Agus\AppData\Local\Microsoft\Windows\INetCacheContent.Word\computer_30842.png

* 1. En un navegador abrir las DevTools o Herramientas de Desarrollo presionando F12. Navegue a <https://tienda.claro.com.ar> y en la pestaña de **Network** o **Red** analice las siguientes peticiones:
* contentManagement?content=Productos\_destacados\_spot
* contentManagement?content=Footer\_Content&page=home
  + 1. De la petición debe obtener:
       1. Método
       2. dirección url
       3. cabecera
       4. cuerpo
    2. De la respuesta debe obtener:
       1. Código de estado
       2. Mensaje de estado
       3. cabeceras
       4. cuerpos
    3. ¿Qué relación guarda el contenido de la respuesta con lo que se observa en la UI?
  1. Acceda al siguiente swagger: <https://petstore.swagger.io/> y verifique el funcionamiento de:
     1. **GET /pet/{petId}**
        1. ¿Qué respuesta ofrece con un petId=1?
        2. ¿Qué respuesta ofrece con un petId=-23 ó petId=23432432?
     2. **DELETE /pet/{petId}**
        1. ¿Qué respuesta ofrece con un petId=2? ¿Y si se intenta realizar esta operación una segunda vez?
     3. **POST /pet/{petId}**
        1. ¿Qué respuesta ofrece con un petId=2 name=CatNew status=available?
  2. Abra postman y siga los siguientes pasos:
     1. Cree una **colección** nueva con el nombre “Pets”
     2. Cree un http
     3. Seleccione GET <https://petstore.swagger.io/v2/pet/4>
     4. Ejecutar. ¿La respuesta es la misma que en swagger?
  3. Abra postman y siga los siguientes pasos:
     1. Cree una **colección** nueva con el nombre “Claro”
     2. En el navegador, en la pestaña de network hacer click en los servicios de productos destacados o del footer. Click cn copy > copy curl as curl (bash)
     3. En postman click en “Import”
     4. Pegar la información copiada y click en “Import into collection”
     5. Ejecutar. ¿La respuesta es la misma que en el navegador?
  4. Supongamos que deseamos diseñar la respuesta de un servicio que es empleado para brindarle información a una página sobre adopción de animales. ¿Qué información debe poseer este JSON?

**Para aprender más...**

* 1. ¿Qué son los proxies y qué función cumplen?
  2. ¿Qué elementos puede contener un servidor?
  3. ¿Qué tipos de recursos se emplean en http?
  4. ¿Qué significa MIME Types?

UNIDAD II: Control de versiones

El control de versiones es una práctica fundamental en el desarrollo de software que permite gestionar y rastrear los cambios en el código fuente a lo largo del tiempo. Existen diversas herramientas diseñadas para facilitar esta tarea, ofreciendo funcionalidades que van desde el control individual hasta la colaboración en grandes equipos de desarrollo.

# TRABAJO PRÁCTICO Nº 2 Control de Versiones

**Objetivos de Aprendizaje**

* Conocer las prácticas de versionado avanzadas, relacionada con el uso de ramas.
* Conocer los estándares para la realización de commits.

**Unidad temática que incluye este trabajo práctico**

Este trabajo práctico corresponde a la unidad Nº 2 de la programación de la asignatura.

**Consignas a desarrollar en el trabajo práctico**

### Preguntas

* 1. ¿Qué es control de versiones?

El **control de versiones** es un sistema que permite rastrear y gestionar los cambios en archivos a lo largo del tiempo. Se usa principalmente en el desarrollo de software para coordinar el trabajo en equipo y mantener un historial de modificaciones.

Beneficios:

* Permite volver a versiones anteriores si algo sale mal
* Facilita la colaboración entre varios desarrolladores.
* Mantiene un historial detallado de cambios y quién los hizo.
* Mantiene un historial detallado de cambios y quién los hizo.
  1. ¿Qué es Git? ¿Quién lo desarrolló? Detalle sus principales características.
  2. Liste los comandos principales de git y explique brevemente para qué sirven.
  3. Escriba los comandos de git correspondientes en cada flecha:
  4. ¿Qué es el modelo de branching?
  5. ¿Qué buenas prácticas se deben seguir a la hora de emplear git?
  6. ¿Qué son los “Conventional Commits”? Liste las ventajas de aplicarlo.
  7. ¿Qué es github?



Para resolver los ejercicios debes contar con: [Git](https://git-scm.com/)



### Ejercicios

* 1. Desde la consola de comandos, resuelva el siguiente ejercicio:

\*Recuerde instalar git en su computadora. Puede descargarlo de https://git-scm.com/

Proyecto Local:

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8 | Inicializar un repositorio  Visualizar las branchs existentes  Crear un archivo y guardarlo\*  Mostrar situación actual del repositorio  Registrar los cambios (añadir el archivo)  Incluir el archivo en el repositorio local  (emplear **conventional commits**)  Visualizar las branchs existentes  Crear una nueva branch a partir de la actual  Moverse a la nueva rama | git  \_ \_ \_ \_  git \_ \_ \_ \_ \_ \_  (manualmente)  git \_ \_ \_ \_ \_ \_  git \_ \_ \_ \_\_\_\_\_\_\_\_  git \_ \_ \_ \_ \_ \_ \_ \_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  git \_ \_ \_ \_ \_ \_  git \_ \_ \_ \_ \_ \_ \_\_\_\_\_\_\_\_\_  git \_ \_ \_ \_ \_ \_ \_ \_ |

\*El archivo creado debe ser **.go** y su contenido puede ser copiado de las diapositivas de clase.

* 1. Hacer grupo de 2 personas. Entrar a <https://github.com/>  y aceptar el proyecto <https://classroom.github.com/a/imdgYQQs>. Configurar el repositorio para trabajar de manera colaborativa con un compañero. (Setting -> Collaborators). En este repositorio
     1. Cree una nueva rama
     2. Vea el listado de todas las **branchs** existentes en el proyecto y la branch actual
     3. Cambie de rama
     4. Realice algún cambio en el código y push al repositorio remoto
     5. Visualice en Github los commits de las **branchs**
     6. Merge de la nueva rama con Master
     7. Borrar una **branch**

**Recomendación:** Puedes utilizar alguna plataforma como Test Automation University para hacer el curso de [Source Control](https://testautomationu.applitools.com/git-tutorial/) (gratuito), DevTalles para [Git+GitHub](https://cursos.devtalles.com/courses/git-github-control-versiones). También puedes buscar cursos en otras plataformas como Coder House, Udemy, Codecademy, Digital House, Code School o Platzi para aprender y practicar sobre Git.

**Para aprender más…**

* 1. ¿Qué otros sistemas de versionado se emplean para compartir código además de Git?

UNIDAD III: Go

Golang es un lenguaje de programación diseñado con el objetivo de ser simple, eficiente y escalable, combinando la facilidad de uso de lenguajes como Python con el alto rendimiento y concurrencia de C o C++.

Go se destaca por su rapidez de compilación, su sólido sistema de concurrencia basado en goroutines y su gestión automática de memoria con recolección de basura. Además, cuenta con un ecosistema robusto y una sintaxis clara que lo hacen ideal para el desarrollo de aplicaciones escalables, sistemas distribuidos y servicios en la nube.

# TRABAJO PRÁCTICO Nº 3 Introducción a Go

**Objetivos de Aprendizaje**

* Adquirir los conocimientos básicos de instalación, configuración y uso del lenguaje go
* Creación de APIs en Go

**Unidad temática que incluye este trabajo práctico**

Este trabajo práctico corresponde a la unidad Nº3 de la programación de la asignatura.

**Consignas a desarrollar en el trabajo práctico**

En la página oficial de **Go** <https://go.dev/learn/#featured-books> podrás encontrar muchos libros y material de ayuda.

### Preguntas:

* 1. ¿Qué es Go o Golang? ¿Qué características tiene?
  2. ¿Cuáles son sus principales ventajas? ¿Qué desventajas posee?
  3. ¿Qué tipos de datos se pueden emplear en go?
  4. ¿Qué son los paquetes?
  5. ¿Qué son los módulos?
  6. ¿Cómo maneja la concurrencia Go?

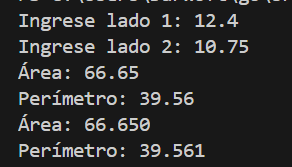
### 

### Ejercicios Prácticos:

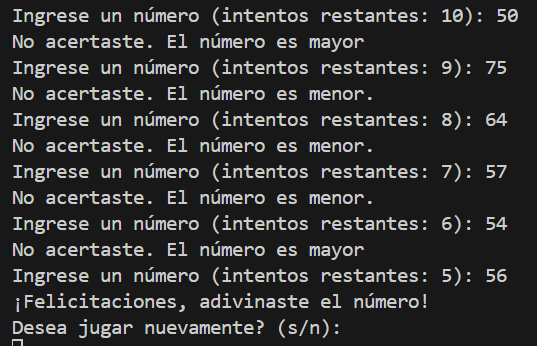
**Triángulo:**

* 1. Crear un programa que solicite al usuario que ingrese los lados de un triángulo rectángulo y luego calcule e imprima el **área** y el **perímetro** del triángulo. El programa debe:
     1. Solicitar al usuario que ingrese la longitud de los dos lados del triángulo rectángulo.
     2. Calcular la hipotenusa del triángulo usando el teorema de Pitágoras.
     3. Calcular el área del triángulo usando la fórmula base x altura / 2.
     4. Calcular el perímetro del triángulo sumando los lados.
     5. Imprimir el área y el perímetro del triángulo con dos decimales de precisión.
     6. El programa debe usar variables para almacenar los lados del triángulo, la hipotenusa, el área y el perímetro. También debe usar constantes para representar el número de decimales de precisión que se desean en la salida.

Ejemplo de salida de pantalla:



**Juego Números Aleatorios:**

* 1. Crear un programa que permita a un usuario adivinar un número. El programa debe:
     1. Generar un número aleatorio entre 0 y 100.
     2. El usuario tiene hasta 10 intentos para acertar el número.
     3. Tras cada intento el programa debe imprimir si el número aleatorio es menor o mayor al número ingresado por el usuario.
     4. Se debe mostrar un menú para permitir jugar nuevamente o salir del programa.

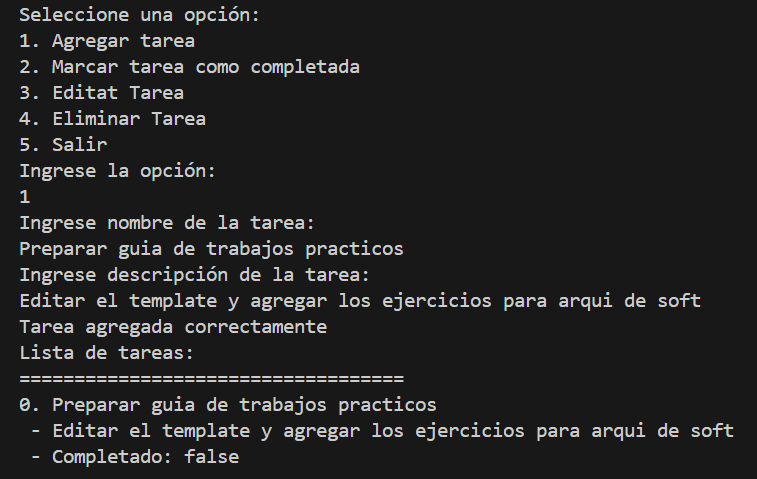
**División:**

* 1. Implementar en un archivo **main.go** una función que calcule la división de 2 números (pueden ser enteros o decimales float). La función debe retornar una variable de tipo **error** si el divisor es igual a cero. Este error debe ser capturado por la función main y mostrar un mensaje por la terminal. En caso contrario, (happy path), debe mostrar el resultado obtenido de la división.
  2. Implementar 2 casos de prueba para esta función validando assertion y validando el caso de error.

**Lista de Tareas:**

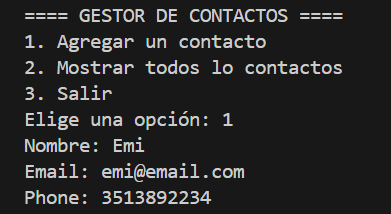
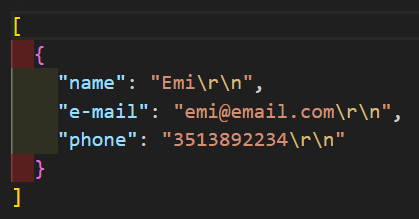
* 1. Crea un programa que permita gestionar una lista de tareas. El programa debe tener en cuenta:
     1. Cada tarea debe tener un título, descripción y si está completada o no (**struc**).
     2. El programa debe permitir agregar, eliminar y actualizar tareas (**slice**)

Ejemplo de salida de pantalla:

****

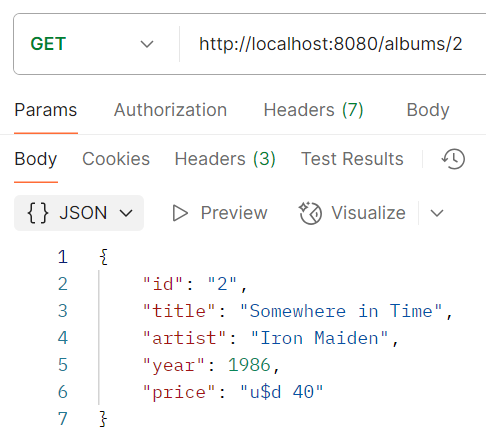
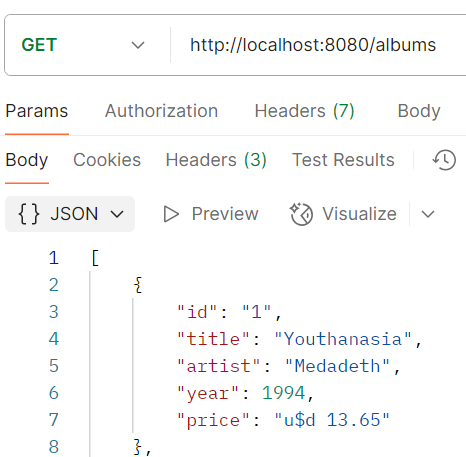
**Gestor de Contactos:**

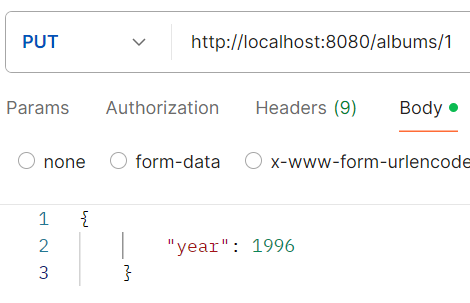
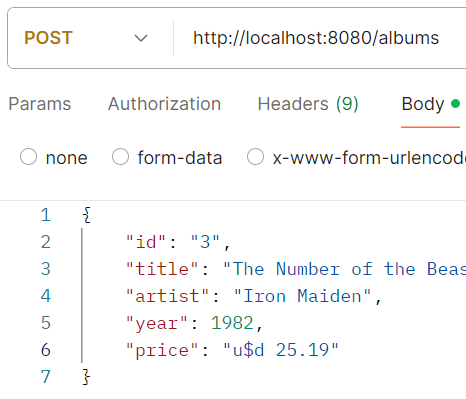
* 1. El programa debe permitir agregar contactos con los datos de nombre, email y teléfono (**struc**). También debe permitir ver todos los contactos. Los mismos se almacenarán en un archivo json. Además, el programa debe hacer manejo de errores.

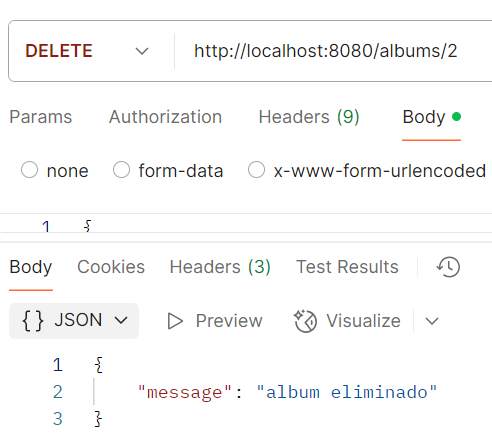
Ejemplo de salida de pantalla y json generado:

**API RestFul de Vinilos:**

* 1. El programa debe proporcionar el acceso a una tienda de vinilos. Se deben crear los endpoints para:
     1. Visualizar el listado de todos los álbumes. **GET /albums**
     2. Agregar álbumes. **POST /albums**
     3. Obtener los datos de un álbum específico. **GET /albums/:id**
     4. Actualizar los datos de un álbum específico. **PUT /albums/:id**
     5. Eliminar un de un álbum específico. **DELETE /albums/:id**

Ejemplo de funcionamiento de los endpoints:

****

****

**Piedra Papel o Tijera:**

* 1. Desarrollar un proyecto web que permita jugar a Piedra-Papel-Tijera contra la computadora. Se debe tener en cuenta:
     1. Renderizado de plantillas HTML y envío de datos
     2. Enrutamiento para poder redireccionar al usuario a través de las diferentes páginas
     3. Los handlers, necesarios para iniciar el juego, comenzar un juego nuevo, renderizar los templates de HTML, etc.

|  |  |
| --- | --- |
| /index | /game |
| /new | /about |

**HTTP:**

* 1. Implementar en un archivo **main.go** una función que se encargue de realizar una llamada a una API de MercadoLibre y muestre los resultados en la terminal (puede ser la API de categorías o la API de search). Para esto se deben utilizar las librerías net/http, fmt, io/ioutil y encoding/json. En caso de obtener un error (código de estado distinto a 200 OK) se debe mostrar el mismo por la terminal.
  2. Implementar un test con assertions de la función que parsea el contenido de los bytes.

**APIS:**

* 1. Crear un proyecto ej-apis
     1. Elegir una API de <https://github.com/public-apis/public-apis>
     2. Crear un branch por cada persona del grupo
     3. Por cada uno implementar un endpoint de la API
     4. Por cada uno implementar 2 test cases

**Ejercicio JWT:**

* 1. Tomar como referencia el proyecto ej-auth: <https://github.com/emikohmann/arq-software/tree/master/ej-auth>
     1. Modificarlo (no es necesario crear un proyecto nuevo) para agregar al engine un nuevo endpoint. El recurso debe estar compuesto de la siguiente manera:
        1. Método: POST
        2. URL: /login-jwt
        3. Crear una nueva función Login JWT en el controller.
        4. Crear una nueva función Login JWT en el servicio.
        5. El servicio debe retornar un token pero esta vez no debe leerlo de un archivo sino generarlo utilizando JWT.



**Para aprender más…**

* 1. ¿Qué diferencia tienen las funciones variádicas, recursivas, anónimas y de orden superior?
  2. ¿Qué son los closures en Go?

UNIDAD IV: Capa de Datos - ORM

La gestión de datos en las aplicaciones modernas no sería tan eficiente sin el uso de una Capa de Datos y ORM (Mapeo Objeto-Relacional). Un ORM nos permite interactuar con bases de datos utilizando código en lugar de consultas SQL manuales, facilitando el desarrollo y mantenimiento de aplicaciones.

Gracias a los ORM, los desarrolladores pueden trabajar con datos de manera más intuitiva, utilizando objetos y relaciones en lugar de tablas y consultas complejas. Hoy en día, estos sistemas no solo se limitan a bases de datos tradicionales, sino que también permiten integraciones con diversas tecnologías y servicios en la nube, mejorando la escalabilidad y el rendimiento de las aplicaciones.

# TRABAJO PRÁCTICO Nº 4 Capa de Datos - ORM

**Objetivos de Aprendizaje**

* Comprender los conceptos fundamentales del Mapeo Objeto-Relacional (ORM) y su importancia en el desarrollo de aplicaciones.
* Adquirir habilidades para interactuar con bases de datos utilizando un ORM en lugar de consultas SQL manuales.

**Unidad temática que incluye este trabajo práctico**

Este trabajo práctico corresponde a la unidad Nº 4 de la programación de la asignatura.

**Consignas a desarrollar en el trabajo práctico**

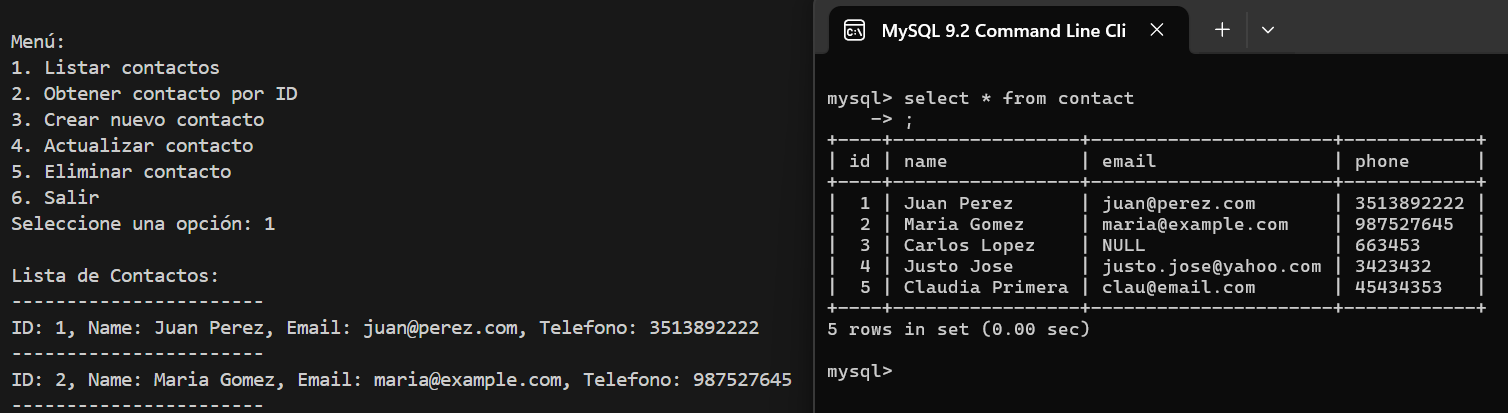
### Preguntas

* 1. ¿Qué es un ORM y cuál es su propósito en el desarrollo de aplicaciones?
  2. ¿Cuáles son las ventajas y desventajas de utilizar un ORM en comparación con escribir consultas SQL manuales?
  3. ¿Cómo se relaciona un ORM con el modelo de base de datos relacional?

### Ejercicios Prácticos:

**Gestor de Contactos con MySQL:**

* 1. El programa debe permitir agregar contactos con los datos de nombre, email y teléfono (**struc**). También debe permitir ver todos los contactos. Se debe tener en cuenta:
     1. Los contactos se almacenarán en una base de datos MySQL.
     2. El programa debe hacer manejo de errores.
     3. En un .env se deben almacenar los datos de la base de datos: DB\_NAME, DB\_USER, DB\_PASSWORD, DB\_HOST y DB\_PORT.



**Recomendación:** Si quieres profundizar en el tema puedes utilizar alguna plataforma como Udemy, Codecademy, Digital House, Code School para aprender y practicar lenguajes como Go.

**Para aprender más…**

* 1. ¿Qué son las migraciones en un ORM? ¿Para qué sirven?
  2. ¿Qué diferencia hay entre un ORM basado en Active Record y uno basado en Data Mapper?.
  3. ¿Qué son las consultas ORM “lazy loading” y “eager loading”?

**¡Trabajo de Evaluación!**: Utiliza lo recientemente aprendido en Go para comenzar a generar el proyecto.

REQUISITOS DEL PROYECTO

**Sobre el frontend:**

* Formulario de Login y botón para hacer Logout
* Página de bienvenida se muestra correctamente
* Se pueden buscar cursos correctamente
* Muestra correctamente resultados de búsqueda / No hay resultados
* Se puede inscribir correctamente a un curso
* Se muestra error si no se puede inscribir a un curso (Por ej. ya inscripto)
* Muestra el listado de cursos a los que se encuentra inscripto un usuario
* Formulario para dejar un comentario y puntuación sobre el curso
* Formulario para crear curso sólo visible usuario administrador
* Formulario para editar curso sólo visible usuario administrador
* Permite al administrador subir un archivo relacionado con el curso
* Opción para eliminar un curso sólo visible usuario administrador
* Los errores al crear/editar/eliminar curso se muestran correctamente

**Sobre el Backend:**

* Login recibe las credenciales y genera un token de acceso
* Implementa correctamente el registro de un usuario nuevo
* Manejo de errores (401 Login inválido, 404 curso no encontrado, 400/409 ya inscripto, etc.)
* Implementa correctamente la búsqueda de cursos
* Implementa correctamente la generación de la inscripción
* Implementa correctamente la obtención de cursos a los que está inscripto un user ID
* Implementa correctamente agregar un comentario y puntuación al curso (y retorna esta información en el GET)
* Implementa correctamente el agregado de un archivo relacionado a un curso
* Implementa estructura MVC y respeta las responsabilidades de modelo, vista y controlador.
* Implementa correctamente la separación entre los objetos del modelos (DAOs) y los DTOs
* Implementa correctamente el manejo de errores (no ignora los errores) y códigos de estado
* Las funciones del backend no ignoran los errores (Validan siempre err != nil)

**Sobre la seguridad:**

* La base de datos no almacena la información de la password de forma plana, usa hashing
* Se valida el tipo de usuario vía token para todas las funciones que lo necesiten (administrador)

**Sobre la base de datos:**

* Se conecta correctamente contra la base de datos usando GORM
* Las distintas tablas en la base no duplican la información en más de una tabla

**Sobre docker:**

* Cada componente de la solución se encuentra Dockerizada (Dockerfile, Docker Compose, etc.)
* La solución completa corre correctamente con Docker

UNIDAD V: Testing

El testing es una parte fundamental del desarrollo de software, ya que permite verificar que nuestro código funcione correctamente y se mantenga estable a lo largo del tiempo. Aplicar buenas prácticas en el testing ayuda a detectar errores temprano y mejora la calidad del software.

En Go, el soporte para testing está integrado en el propio lenguaje a través del paquete testing, lo que facilita la escritura y ejecución de pruebas unitarias sin necesidad de herramientas externas.

# TRABAJO PRÁCTICO Nº 5 Testing

**Objetivos de Aprendizaje**

* Comprender los diferentes tipos de pruebas que se pueden realizar en software.
* Adquirir habilidad en el desarrollo de pruebas unitarias, y en la medición de la cobertura.

**Unidad temática que incluye este trabajo práctico**

* Este trabajo práctico corresponde a la unidad Nº 5 de la programación de la asignatura.

**Consignas a desarrollar en el trabajo práctico**

### Preguntas

* 1. ¿Por qué es útil contar con pruebas de software?
  2. ¿Cómo se pueden clasificar las pruebas? Explique cada tipo de prueba.
  3. ¿Qué es la pirámide de pruebas?
  4. ¿Qué son las pruebas de caja blanca y de caja negra?
  5. ¿A qué se refiere la cobertura de las pruebas?



### Ejercicios Prácticos:

* 1. Cree una función básica de suma que devuelva a+b. Escriba pruebas unitarias para darle cobertura a esta funcionalidad.
  2. Cree una función básica que retorne el máximo número de 2 números enviados. Escriba pruebas unitarias para darle cobertura.
  3. Cree una función para el cálculo de fibonacci que sea recursiva. Cree pruebas unitarias para darle cobertura.
  4. Ejecute el comando que permite medir el coverage. Guardar la información obtenida en un documento.

**Para aprender más…**

* 1. ¿Cómo se pueden simular errores o condiciones especiales en pruebas unitarias sin modificar el código fuente?.
  2. ¿Cómo se pueden ejecutar pruebas en paralelo en Go y en qué casos es útil?
  3. ¿Cómo se pueden realizar pruebas con diferentes conjuntos de datos sin repetir código?

UNIDAD VI: Docker

Permite crear, distribuir y ejecutar aplicaciones en contenedores, un método ligero y eficiente para empaquetar software junto con todas sus dependencias. Al compartir el kernel del sistema anfitrión, su ejecución es más rápida y eficiente en el uso de recursos.

Con esta herramienta los desarrolladores pueden asegurarse de que sus aplicaciones se ejecuten de la misma manera en cualquier entorno, ya sea en desarrollo, pruebas o producción. Esto facilita la portabilidad, escalabilidad y despliegue de aplicaciones en la nube o en servidores locales.

# TRABAJO PRÁCTICO Nº 6 Contenedores

**Objetivos de Aprendizaje**

* Adquirir conceptos básicos sobre docker.
* Adquirir habilidad en el despliegue de contenedores.

**Unidad temática que incluye este trabajo práctico**

Este trabajo práctico corresponde a la unidad Nº 6 de la programación de la asignatura.

**Consignas a desarrollar en el trabajo práctico**

Puedes consultar la documentación oficial de docker: <https://docs.docker.com/get-started/> 

### PreguntasC:\Users\Agus\AppData\Local\Microsoft\Windows\INetCacheContent.Word\pregunta.png

* 1. ¿Qué es Docker y para qué se utiliza?
  2. ¿Cuál es la diferencia entre una máquina virtual y un contenedor de Docker?
  3. ¿Qué ventaja ofrece Docker en comparación con la instalación manual de dependencias en un servidor?
  4. ¿Qué es una imagen en Docker y cómo se diferencia de un contenedor?
  5. ¿Qué es un Dockerfile y para qué se utiliza?
  6. ¿Qué es Docker Compose y en qué casos se recomienda su uso?

### 

### Ejercicios Prácticos:

* 1. Ejecute en la terminal los comandos para:
     1. Comprobar los contenedor existentes
     2. Comprobar las imágenes existentes
     3. Crear un contenedor con la imagen **hello-world**
     4. Crear un contenedor con la imagen hello-world y llamarlo hola-mundo
     5. Ejecutar un contenedor
     6. Eliminar un contenedor
  2. Se debe crear un contenedor básico con una aplicación Go.
     1. Escribir un programa en Go que imprima "¡Hola, Docker!".
     2. Crear un Dockerfile para compilar y ejecutar la aplicación dentro de un contenedor.
     3. Construir la imagen con docker build.
     4. Ejecutar el contenedor con docker run.
  3. Se debe crear un servidor Web en Go con Docker.
     1. Escribir un servidor HTTP en Go que escuche en el puerto 8080 y devuelva "¡Bienvenido a mi servidor en Docker!".
     2. Crear un Dockerfile para compilar y ejecutar el servidor dentro de un contenedor.
     3. Exponer el puerto 8080 en Docker para acceder a la aplicación desde el navegador.
     4. Ejecutar la imagen y probar la API con curl o el navegador.
  4. Se debe crear un volúmen en Docker con Go para persistir datos.
     1. Escribir un programa en Go que guarde mensajes en un archivo de texto.
     2. Montar un volumen en Docker para que los datos no se pierdan al reiniciar el contenedor.
     3. Ejecutar el contenedor varias veces y comprobar que los mensajes persisten.
  5. Emplear Docker Compose con Go y una Base de Datos para correr múltiples servicios.
     1. Escribir una API en Go que se conecte a una base de datos PostgreSQL.
     2. Crear un docker-compose.yml para levantar ambos servicios (API y DB).
     3. Probar que la API puede leer y escribir en la base de datos desde un contenedor.



**Para aprender más…**

* 1. ¿Cómo funciona el sistema de archivos en capas de Docker y qué ventaja ofrece?
  2. ¿Qué es un espacio de nombres (namespace) y cómo lo usa Docker para el aislamiento de contenedores?
  3. ¿Qué es un grupo de control (cgroup) y cómo ayuda a gestionar los recursos de los contenedores en Docker?
  4. ¿Cuál es la diferencia entre imágenes "lightweight" (basadas en Alpine) y las basadas en distribuciones completas como Ubuntu?