### Go Basic

Objetivo del proyecto

* Aprender los conceptos básicos para programar en **Go**.
* Mejorar tus conocimientos en **backend**
* Mejorar tus conocimientos sobre crear una **API Rest**

### Analiza la interfaz

Podemos generar un ejecutable en la ruta sabemos que go es un lenguaje tipado

Go build index.go

Para compilarlo seria

./index.exe

### De esta manera manejamios la interfaces de go con las

### variables

Una variable es una caja donde guardamos información lo destinguimos de la siguiente manera con el tipo de datos que seria con int, string, bool, float32, float34

package main

import "fmt"

func main() {

    //*para definir una variable con la palabra var*

    var suma int = 8 + 3

    var resta int = 6 - 5

    var nombre string = "Robert"

    fmt.Println("hola Mundo " + nombre)

    //*Aqui emprimimos la suma*

    fmt.Println(suma)

    fmt.Println(resta)

}

En esta parte también se hace una asignación de variable con dos puntuos igual := en la cual go automáticamente va destinguir que tipo de variable es

//*:= con esta distinción  go va adivinar que variables es*

    pais := "España"

Funciones

Para realizar la función hemos de poner func mas el nombre de la función enviando tres paramentros, con sus tipos de datos

func operacion(n1 float32, n2 float32, op string) float32 {

    var resultado float32

    if op == "+" {

        resultado = n1 + n2

    }

    if op == "-" {

        resultado = n1 - n2

    }

    if op == "\*" {

        resultado = n1 \* n2

    }

    if op == "/" {

        resultado = n1 / n2

    }

    return resultado

}

func calculator(numero1 float32, numero2 float32) {

    fmt.Println("La Suma es:")

    fmt.Println(operacion(numero1, numero2, "+"))

    //*Resta*

    fmt.Println("La Resta es:")

    fmt.Println(operacion(numero1, numero2, "-"))

    //*Multiplicacon*

    fmt.Println("La Multiplicacion es:")

    fmt.Println(operacion(numero1, numero2, "\*"))

    //*Division*

    fmt.Println("La Division es:")

    fmt.Println(operacion(numero1, numero2, "/"))

    Hola()

}

Arrays

En el array solo definimos con los corchetes con sus indice

func main() {

    //*indicamos cuantos elementos va tener el tipo de elementos*

    var movies [3]string

    movies[0] = "La verdad Duele"

    movies[1] = "Ciudadano ejemplar"

    movies[2] = "Gran Torino"

    movies := [3]string{"La verdad Duele", "Ciudadano ejemplar", "Gran Torino"}

    //*lo mostramos*

    fmt.Println(movies[1])

}

Array multidicionales

Array multidimensional

Lo hacemos de la siguiente manera

sacamos toda la collecion del array

ya para mostrarlo los sacamion con la posicion y el índice

var peliculas [3][2]string;

    peliculas[0][0] = "La verdad duele"

    peliculas[0][1] = "Mientras duermes"

    peliculas[1][0] = "Ciudano ejemplar"

    peliculas[1][1] = "El señor de los anillos"

    peliculas[2][0] = "Gran torino"

    peliculas[2][1] = "A todo gas"

SLICE

SLICE no tiene el numero indice marcado es array dinamico no

necesitamos definir el numero de elemento

ejemplo con un append añadimos mas datos con la variable

con len podemos ver la longitud que tiene el array

peliculas := []string{

        "La verdad duele",

        "Ciudadano Ejemplar",

        "Batman",

        "Superman"}

    peliculas = append(peliculas, "Sin limites")

    peliculas = append(peliculas, "Camp Rock")

    peliculas = append(peliculas, "A todo gas")

    fmt.Println(peliculas[0:3])

Punteros

Para declarar un apuntador en Go se utiliza la siguiente sintaxis:

TipoApuntador = "\*" TipoBase

TipoBase = TipoDato

En la siguiente línea de código se declara un apuntador a un entero:

var apunta\_entero \*int

**Como accedemos a la dirección de una variable**

En la tabla de símbolos todas las variables tienen asignada la dirección de memoria donde se localizan, la cual se puede acceder si le colocamos el operador ampersand (*&*) al inicio de ellas. En el siguiente ejemplo puedes observar dicho operador siendo utilizado para imprimir la dirección de una variable:

func main() {

    //*punteros almacenar en un asterisco*

    var x, y \*int

    //*almacenando los datos en otra variable*

    entero := 5

    //*& con el valor amperson indicamos que accedamos a la direccion de memoria no al valor en si*

    x = &entero

    y = &entero

    //*estamos colocando el asterisco adelante para que muestre el valor si no colocamos vamos a l a direccino*

    \*x = 6

    fmt.Println(\*x)

    fmt.Println(\*y)

}

Condicional IF

Cuando necesitamos que el programa ejecute ciertas sentencias cuando la condición es verdadera, pero a su vez necesitamos que ejecute otras sentencias solamente cuando la condición es falsa, necesitamos una sentencia condicional ***if-else***.

func main() {

  /\**variable local de tipo entero*\*/

  var calificacion int = 5

  /\**Sentencia if, que verifica si calificación es menor a 6*\*/

  if calificacion < 6 {

    /\**Si la condición se cumple, imprime*\*/

    fmt.Println("Reprobaste")

  }

  fmt.Println("Tu calificación fue de: ", calificacion)

}

Switch

La sentencia ***switch*** permite evaluar una variable de prueba contra una lista de valores conocidos como casos (***case***). En Go un ***switch*** puede ser de dos tipos posible:

* **Expresión** – Los casos contienen expresiones que son comparadas con el valor de la variable de prueba.

**Tipo** – Los casos contienen un tipo de dato que es comparado con el tipo de dato de la variable de prueba. package main

import "fmt"

func main() {

    /\**variable local de tipo entero*\*/

    var hora int = 10

    /\**Se pasa hora como variable de prueba*\*/

    switch hora {

    /\**Si hora coincide con alguna de las literales especificadas*\*/

    case 1, 2, 3, 4:

        fmt.Println("Aún es temprano")

    case 5, 6, 7:

        fmt.Println("Está atardeciendo")

    case 8:

        fmt.Println("Acaba de oscurecer")

    case 9, 10, 11:

        fmt.Println("Ya es tarde")

    default:

        fmt.Println("Es demasiado tarde")

    }

}

Sentencia FOR

La sentencia ***for*** controla una sección de código para que se repita una cierta cantidad de veces especificada al escribir el programa o determinada en tiempo de ejecución. La sintaxis es la siguiente

Go cuenta con una sentencia ***break*** cuyos usos son terminar el ciclo más interno en el que esté situado, o bien terminar un case de un switch

package main

import "fmt"

func main(){

  for i:=0 ; i < 10; i++ {

    fmt.Printf("Valor de i: %d", i)

    if i == 7{

      fmt.Printf(" así que saldremos del ciclo...\n")

      break

    }

    fmt.Printf("\n")

  }

}

For Each

Aquí recorremos el array saldría los mismo valores pero con menos codigo

*peliculas := []string{"Pelicula 1", "El club de la lucha", "A todo gas", "Gran torino"}*

*for \_, pelicula := range peliculas {*

*fmt.Println(pelicula);*

*}*

STRUCT

Se define el tipo de datos personalizado para definirlo como un objeto

indicando con los tipos de modelo pasandole unos parametros ya definidos

type Gorra struct {

    marca  string

    color  string

    precio float32

    plana  bool

}

func main() {

    var gorra\_negra = Gorra{"Adidas", "negra", 18.50, true}

    fmt.Println(gorra\_negra)

}

Puede usar datos de cuerpo sin procesar para enviar cualquier cosa que pueda ingresar como texto. Use la pestaña sin **procesar** y la lista desplegable de tipos para indicar el formato de sus datos ( **Texto** , **JavaScript** , **JSON** , **HTML** o **XML** ) y Postman habilitará el resaltado de sintaxis y agregará los encabezados relevantes a su solicitud.

Maps

Nos permite almacenar información de forma similar como lo hacíamos con los arrays o los slice, pero con un map tiene la particularidad tiene asociada una llave.

1. Mapa = "map" "[" TipoLlave "]" TipoElemento
2. TipoLlave = Tipo

En el siguiente ejemplo declaro un **map** que almacenará días de la semana donde la llave única será un número entero (correspondiente al orden de los días) y el elemento principal (el que no es la llave única) será una cadena de texto:

diasSemana = make(map[int]string)

Para insertar elementos en el **map** anterior (uno a uno):

1. diasSemana[1] = "Domingo"
2. diasSemana[2] = "Lunes"
3. diasSemana[3] = "Martes"
4. diasSemana[4] = "Miércoles"
5. diasSemana[5] = "Jueves"
6. diasSemana[6] = "Viernes"
7. diasSemana[7] = "Sábado"

Para recorrer todos los elementos debemos ser los siguientes

for dia := range diasSemana {

fmt.Println("El día", dia, "de la semana es", diasSemana[dia])

}

**Para buscar un elemento es necesario utilizar una asignación doble**

En donde la primera variable almacenará el resultado de la consulta y la segunda variable (generalmente llamada *ok*<) será inicializada con un valor *bool true* si el elemento existe:

1. string\_dia, ok := diasSemana[8]
2. **if**(ok){
3. fmt.Println("El elemento sí existe y almacena:", string\_dia)
4. }**else**{
5. fmt.Println("El elemento no existe")
6. }

Para consultar el tamaño de un map se utiliza la función **len()** como lo hacíamos con *slices*:

len(diasSemana)

La función **delete()** permite eliminar elementos de un **map** a través de la llave única de estos:

**delete**(diasSemana, 2) //Elimina el elemento con la llave 2

INTERFACE

ERROR

Practica teorica

Una vez completada la parte teórica pasarás a realizar la práctica. Deberás desarrollar una API Rest que sea capaz de recibir las siguientes peticiones y enviar una respuesta a cada una de ellas:

* Petición que devuelva solo los siguientes headers:
  + HTTP status code
  + Un header personalizado
    - Ejemplo: Server → A Go Web Server
* Petición que devuelva una plantilla HTML
  + Puedes usar cualquier código HTML, lo importante es que sepas cómo hacer que la API devuelva la petición **desde un fichero HTML**
* Petición que devuelva XML
  + La API deberá devolver un objeto que contenga la siguiente estructura:

<Profile>

<Name>Your name</Name>

<Hobbies>

<Hobby>Hobby 1</Hobby>

<Hobby>Hobby 2</Hobby>

</Hobbies>

</Profile>

* Petición que devuelva **JSON**
  + La API deberá devolver un objeto que contenga la estructura usada anteriormente pero en lenguaje **json**
* Petición que devuelva un **texto plano**
  + La API deberá devolver información a tu elección en texto plano
* Petición que devuelva una imagen
  + La API deberá devolver una imagen a tu elección obtenida a través de un fichero

Collection

Requisitos del proyecto

API de ejemplo: <http://dummy.restapiexample.com/>

* Crea una nueva Collection de **POSTMAN** ( investiga previamente sobre ello )
* Crea una petición para cada uno de los siguientes métodos tal y como muestra la imagen superior:
  + **GET**
  + **POST**
  + **PUT**
  + **DELETE**

La idea principal es que sepas de que forma usar cada tipo de petición y poder así testear Aplicaciones propias o de terceros. Por ejemplo, para la petición que haga uso del método **GET**llamarías a “/employee” para la de **POST** “/create”, …

* Crea un **testing**para cada petición anterior donde comprobarás que la respuesta cumpla con las siguientes condiciones:
  + El código del estado es el esperado
  + El nombre del código del estado es el correspondiente (OK, Created, …)
  + La respuesta tiene **body**
  + La respuesta es en **JSON**
  + El header tiene “**Content-Type**”
  + El tiempo de respuesta es inferior a **200ms**
* Crea un **monitor**que **ejecute cada 5 min** la colección creada anteriormente
* Crea un **documento API** de la colección anterior con la aplicación de Postman
* **Exporta la colección** creada e importala de nuevo en Postman (deberás incorporar el fichero generado en el repositorio del proyecto.

Deberás documentar todas las acciones realizadas anteriormente y realizar capturas de pantalla de todos los procesos.

Este interfaz solo debe realizarse si el usuario esta logeado a parte administrativo

Es importante que tengas en cuenta los roles de usuario ya que un usuario **no logeado** no debería poder realizar las acciones de un usuario **logeado**.

También desarrollarás un panel de control para poder gestionar los los about que cubrirá las siguientes necesidades Backend:

1. Hacer login para identifacarse panel de control.
2. Hacer logout.
3. Modificar la información de resumen sobre ti (About)
4. CRUD de las tecnologías
5. CRUD de las experiencias
6. CRUD de las soft skills
7. Listar peticiones de contacto que se han realizado

Implementación del proyecto

Para llevar a cabo este proyecto será necesario que tengas en cuenta dos roles principales.

* El sistema de **backend**que se encarga de obtener la información de la base de datos
* El sistema de **frontend** que se encargará de consumir lo datos que provee tu propio sistema de backend.
* Es importante que tengas en cuenta que existen dos “Frontales”:
* El portfolio público
* El panel de control que se usará para gestionar la información del porfolio

**La Arquitectura**

Es importante que la lógica de la aplicación (el dominio) esté totalmente aislado del resto de la aplicación y que uses el patrón de arquitectura **MVC**. Este patrón te obligará a organizar tu código de forma limpia y clara.

## Especificaciones del proyecto

1. Debes de usar GIT desde el inicio del proyecto
2. Debes desarrollar el proyecto con **NodeJS**y **ExpressJS**
3. Deberás usar **MySql**para gestionar la base de datos.
4. Todo el código debe de estar debidamente documentado
5. Debes de incluir una pequeña guía de usuario para entender cómo interactuar con la herramienta
6. La interfaz debe de ser totalmente **responsive**
7. Todos los comentarios incluidos en el código tienen que estar escritos en inglés
8. Usa el estilo de código camelCase
9. En el caso de estar usando HTML nunca uses estilos en línea
10. En el caso de estar usando diferentes lenguajes de programación siempre define la implementación en términos separados
11. Es recomendado dividir las tareas en varias subtareas para que de esta forma puedas asociar cada paso en particular de la construcción con un commit específico
12. Para la documentación del proyecto es requerida una versión en PDF dentro del repositorio
13. Debes intentar en la medida de lo posible que los commits y las tareas planificadas sean lo mismo
14. Borra los ficheros que no se usen
15. Intenta usar un Linter de PHP para que tu código sea lo más conciso y estándar posible



|  |  |  |  |
| --- | --- | --- | --- |
| **Tarea** | **Prioridad** | **Horas** | **Dificultad** |
| Documentación | Alta | 2,00 | Alta |
| Organización | Alta | 2,00 | Alta |
| Búsqueda Previa de información | Normal | 2,00 | Normal |
| Creación de repositorio | Baja | 0,15 | Baja |
| Documentando Proyecto | Normal | 4,00 | Normal |
| Estructura index.js | Normal | 1,30 | normal |
| Template engine (pug) Routes | Alta | 2,00 | Alta |
| Arquitectura MVC | Alta | 2,00 | Normal |
| Views layout | Normal | 1:00 | Normal |
| Index.pug | Normal | 1,00 | Normal |
| Controllers | Alta | 4.00 | Alta |
| BD conect | Alta | 1.00 | Normal |
| Models | Alta | 4.00 | Alta |
| Crud por sesión 2 horas | Alta | 6. horas |  |
| Creación README | Baja | 2,00 | Baja |
| Testing / Corrección Errores | Alta | 2,00 | Alta |

Tenemos previsto terminar el proyecto en 32 Horas

En las cual damos 8 horas más para prever incidencias en el proyecto

Registro De Incidencia que se ha detectado durante el proyecto

* Petición a la base de datos solución hay varias petición como hacerlo
* Controlador llamando al model solución con un requiere\_once del model
* Controllador enviando datos a views correctamente solución petición desde views desde action con el controlador mas el método
* En rutamiento de controller con routes
* Crud por seccion
* Login

Seguimiento Calendario del proyecto

Métricas de Calidad

A pesar de que el proyecto debe adherirse a todos los [*requisitos*](https://docs.google.com/document/d/1dtnHybW7qhBAuuuNEZi1Pb0neJEqbME-hXvGSw2XHUI/edit#heading=h.rf0k2roeckt1) y [*especificaciones*](https://docs.google.com/document/d/1dtnHybW7qhBAuuuNEZi1Pb0neJEqbME-hXvGSw2XHUI/edit#heading=h.xr3uz2kyop9d)del proyecto, hay algunas condiciones que, si se cumplen adecuadamente, añaden una sensación de calidad y robustez al propio proyecto. Estas condiciones son:

1. Levantar el npm server conectándose a mysql y localhost.
2. El código NodeJS utilizamos Visual Studio Code
3. El código NodeJS debe estar libre de pelusas.
4. La aplicación web debe responder.
5. La aplicación web debe ser compatible con los principales navegadores del mercado:
   * Internet Explorer 11 o superior.
   * Safari en una de sus últimas versiones.
   * Firefox en una de sus últimas versiones.
   * Chrome en una de sus últimas versiones.

Documentación de Riesgos

* Retrasos de proyecto.
* Pérdida o daños de material de trabajo.

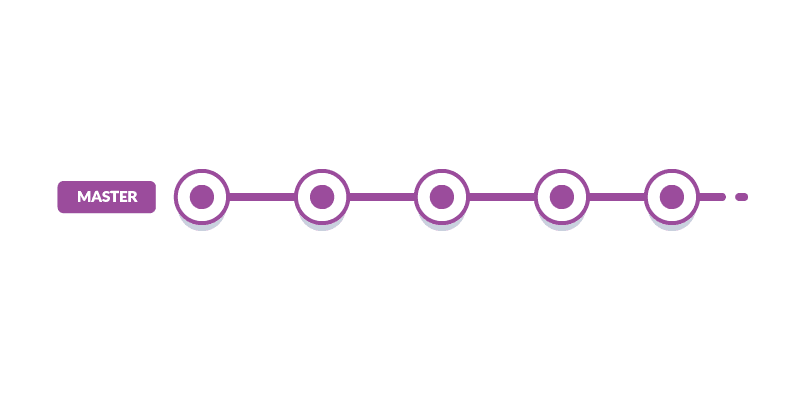
Documentación WORKFLOW DE GIT

* Creación Git Hub https://github.com/robertfox11/NodeJS\_PortFolio.git
* Hacemos commits de la estructura de la página principal.
* Probabilidad de que ocurra 80%
* Impacto en el proyecto 60%
* Posible alternativa (mitigación) Pedir ayuda a compañeros
* Probabilidad de que ocurra 30%
* Impacto en el proyecto 60%
* Posible alternativa (mitigación) Pedir ayuda a compañeros
* No encontrar con facilidad la información relacionada con el proyecto
* Probabilidad de que ocurra 30%
* Impacto en el proyecto 60%
* Posible alternativa (mitigación)
* Pedir ayuda a compañeros

A partir de la realización de la estructura se continuó trabajando solamente en la

rama “master”, a través del Workflow “Gitflow”.

Mas información --> <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow>



Herramiento del proyecto

Se utilizaron diferentes herramientas en el desarrollo del proyecto. Son los siguientes:

1. ***git: un potente sistema de control de*** versiones que ayuda a realizar un seguimiento de los cambios en el árbol de trabajo.
2. ***Visual Studio Code: un editor de*** código optimizado para crear y depurar aplicaciones web modernas.
3. ***NodJS, para realizar la el proyecto con el framework express***
4. ***Npm, para descargar librerias***
5. ***Herramientas para desarrolladores de Google Chrome:*** se utiliza para depurar el código JavaScript y para probar los ajustes de diseño.
6. ***Documentos de Google:*** se utiliza para escribir la documentación del proyecto.
7. [***Validador W3C***](https://validator.w3.org/)***:*** utilizado para validar el código HTML y CSS.
8. [***ESLint***](https://eslint.org/demo)***:*** utilizado para validar el código JavaScript.

## Flujo de trabajo de Git

Todas las confirmaciones se van a insertar en la rama ***maestra,*** siguiendo un criterio personal de cargar solo instantáneas que son funcionales y que funcionan correctamente, sin contar errores menores. No hay otras ramas, ya que ralentizaría el proceso de desarrollo.

En otro lado, los mensajes de confirmación finalizan con su objetivo principal indicado entre corchetes: por ejemplo.

* Utilizaremos un template index.pug que las instalaremos con npm
* Las relaciones principal para la logica funcional es [class]
* las confirmaciones relacionadas principalmente con los cambios CSS comienzan con ***[styling]***;
* los relacionados principalmente con el diseño de la página, ***[layout]***; los
* Los relacionado con el proyecto añadiendo carpetas o documentacion, plugins, library ***[us]****;*
* relacionados principalmente con la documentación, ***[documentation]***.
* Para NodeJS vista la relacionamos con [Proyecto]
* Esctructura mvc

## Estructura del archivo

Los archivos del proyecto se organizarán de esta manera:

ProyectoNodeJS/

public/

img/ Carpeta que contiene todas las imágenes utilizadas en la interfaz.

js/ Carpeta que contiene todos los scripts utilizados en la interfaz de usuario.

css/ Carpeta que contiene todos los estilos utilizados en la interfaz de usuario.

Config/config.php

Configuración de en rutamiento principal

DataBase.php

Petición de conexión a la base de datos MySql

Controller/

Controladores al modelo y al views

Documentation/

Documentation del proyecto

Helpers/ Una funcionalida de vardump para ver la peticiones

Views/

Vistas de rutas para la acción de los métodos que viene de controllers

/layout

Vistas principales de header footer y aside de pug

.gitignore Carpeta utilizada por *git* para contener información sobre el repositorio.

portafoliobd Usaremos para realizar las peticiones

index.js Página principal de la aplicación web.

README.mdArchivo que contiene instrucciones sobre cómo ejecutar el proyecto.

Registro de lecciones aprendidas.

* Realización de una mejora de la documentación más detallada
* Estructura de la clase
* Conceptos MVC, para tener un código estructurado para un petición mas rápida del cliente.
* Petición a la base de datos con **sequelize**
* NodeJS levantar la aplicación con npm server
* El funcionamiento del Framework Express
* El funcionamiento del Framework Routes
* Para utilizar la console.log con bodyparser
* Session para conexión del login
* Passport para initializar la sessión