UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

# Computación Tolerante a Fallas

# *Proyecto Final: Blockchain*

Aceves Curiel Georgina Guadalupe
Fernández Flores Paola Crineth
Preciado Antón Hanna Lizette

# INTRODUCCIÓN

El propósito de este programa es implementar blockchain a la aplicación que desarrollemos como equipo, esto para hacerla una aplicación descentralizada (DApp) las cuales tienen como objetivo utilizar estos bloques de exploración para que los usuarios se relacionen directamente entre ellos y cierren acuerdos sin que exista una entidad central que gestione el servicio.

Las aplicaciones descentralizadas hacen referencia a las plataformas que permiten cualquier interacción entre sus miembros, desde la web o a través de una 'app' móvil, sin la necesidad de un agente central que gestione ese servicio o que lleve un control de cada uno de los registros y acciones realizadas.

Además, ese carácter distribuido propio de una cadena de bloques garantiza que en el caso de que desaparezca un nodo, existen otras muchas “copias de seguridad”. Esto acarrea lo que algunos expertos ven como el principal punto negativo de las DApps: almacenar una gran cantidad de datos en una cadena de bloques puede resultar costoso y aumenta de forma notable el tamaño de la misma en megabytes.

La descentralización de las aplicaciones tiene su origen en 'blockchain', la tecnología que está detrás de la criptomoneda 'bitcoin'. Si bien en un principio estas cadenas de bloques se utilizaron para establecer un medio de pago entre iguales, sin intermediarios, pronto se vio que esa forma de encriptar las transacciones podía tener otras utilidades si sobre ella se permitía la ejecución de otros programas.

## DESARROLLO

Para poder realizar este programa nosotras desarrollamos un programa usando el lenguaje de programación Golang, este programa utiliza una estructura con id, nombre y un map de materias el cual contiene otro map para agregar las materias de algún curso. Este se construyó con el fin de enfocarlo a ser un sistema descentralizado y con la ayuda de un servidor web.

Un servidor web es aquel que permite conexiones remotas comúnmente desde un navegador por medio del protocolo *http*. Estas peticiones suelen ser acompañadas por dos métodos: GET y POST. Cuando queremos navegar o conectarnos a una dirección web, estamos haciendo una petición o *request* de tipo GET. En el caso de la petición POST desde el navegador, esta suele ser usada para enviar información que fue captura desde un *form* o formulario, para que el servidor la almacene en algún lado, modifique o actualice.

La ventaja de usar esta arquitectura radica en la facilidad de comunicación con clientes desarrollados en diferentes lenguajes de programación. Actualmente esta tecnología es ampliamente utilizada en la mayoría de aplicaciones web y móviles.

A continuación la explicación del código:

En las primeras líneas se encuentran los paquetes necesarios para el desarrollo del código, tenemos las variables correspondientes a los maps en los que se almacenan las materias y los alumnos así como la estructura Alumnos que contiene nombre, materia y calificación, se crea un slice de tipo alumnos que será parte del administrador de Servidor, también se tiene la función que añade un alumno al slice de alumnos.

```go
package main

import (
    "fmt"
    "io/ioutil"
    "net/http"
    "strconv"
)

var subjects map[string]map[string]float64
var students map[string]map[string]float64

type Alumnos struct {
    Name    string
    Subject string
    Grade   string
}

type AdminServer struct {
    Info []Alumnos
}

func (info *AdminServer) Add(data Alumnos) {
    info.Info = append(info.Info, data)
}

var auxStudents AdminServer
```

La función agregarCalificación primero verifica que no exista una calificación para ese estudiante en la materia elegida, si no es así se crean los maps y se confirma la existencia de a: el alumno y b: la materia, si no existe alguno o ninguno se crean para poder añadirlos correctamente.

```
func AddGrade(data []string, reply *string) error {
    if _, err := students[data[0]][data[1]]; err {
        *reply = "Error. Ya se tiene una calificacion registrada"
        return nil
    }
    subjectAux := make(map[string]float64)
    studentAux := make(map[string]float64)
    gradeAux, _ := strconv.ParseFloat(data[2], 64)
    subjectAux[data[1]] = gradeAux
    studentAux[data[0]] = gradeAux

    if _, err := students[data[0]]; err {
        students[data[0]][data[1]] = gradeAux
        if _, err := subjects[data[1]]; err {
            subjects[data[1]][data[0]] = gradeAux
        } else {
            subjects[data[1]] = studentAux
        }
    } else {
        students[data[0]] = subjectAux
        if _, err := subjects[data[1]]; err {
            subjects[data[1]][data[0]] = gradeAux
        } else {
            subjects[data[1]] = studentAux
        }
    }
    *reply = "Se agrego correctamente la calificacion del alumno " + data[0]
    return nil
}
```

Esta función es la que se encarga de sacar el promedio general de un estudiante, tomando como referencia el nombre de este.

```
func AverageGradeStudent(name string, reply *string) error {
    if _, err := students[name]; err {
        var sum float64
        var subject int
        for _, grade := range students[name] {
            sum = sum + grade
            subject = subject + 1
        }
        finalGrade := sum / float64(subject)
        *reply = "La calificacion del estudiante " + name + " es: " + strconv.FormatFloat(finalGrade, 'f', 2, 64)
        return nil
    } else {
        *reply = "Error. No existe ese alumno."
        return nil
    }
}
```

La siguiente función es la que calcula el promedio general de una materia, al igual que la anterior, toma como referencia el nombre de la materia que el usuario proporciona.

```
func AverageGradeSubject(subject string, reply *string) error {
    if _, err := subjects[subject]; err {
        var sum float64
        var counter int
        for _, grade := range subjects[subject] {
            sum = sum + grade
            counter = counter + 1
        }
        finalGrade := sum / float64(counter)
        *reply = "El promedio general de la materia " + subject + " es: " + strconv.FormatFloat(finalGrade, 'f', 2, 64)
        return nil
    } else {
        *reply = "Error. No existe esa materia"
        return nil
    }
}
```

Esta función es la que se encarga de obtener el promedio general, de todos los alumnos en todas las materias.

```
func AverageGradeAll(all int, reply *string) error {
    var sum float64
    var cont int
    for student := range students {
        for _, grade := range students[student] {
            sum = sum + grade
            cont = cont + 1
        }
    }
    finalGrade := sum / float64(cont)
    *reply = "El promedio general es: " + strconv.FormatFloat(finalGrade, 'f', 0, 64)
    return nil
}
```

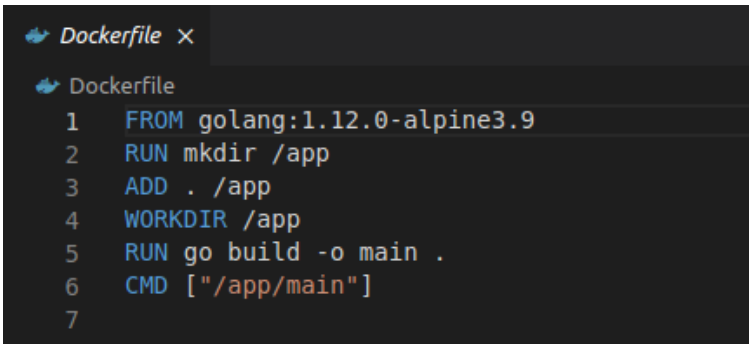
La función form construye el encabezado de el archivo html y cargar html se utiliza para no embeber el código html de esta forma solo se indica que se va a leer el archivo form.
En el main tenemos las rutas que se utilizarán en el navegador para mostrar la información, se utiliza el puerto 9000.

```
func form(res http.ResponseWriter, req *http.Request) {
    res.Header().Set(
        "Content-Type",
        "text/html",
    )
    fmt.Fprintf(
        res,
        cargarHtml("form.html"),
    )
}

func cargarHtml(a string) string {
    html, _ := ioutil.ReadFile(a)
    return string(html)
}

func main() {
    subjects = make(map[string]map[string]float64)
    students = make(map[string]map[string]float64)
    http.HandleFunc("/form", form)
    http.HandleFunc("/alumnos", alumnos)
    http.HandleFunc("/promedioAlumno", alumnos)
    http.HandleFunc("/promedioMateria", alumnos)
    http.HandleFunc("/promedioGeneral", alumnos)
    fmt.Println("Corriendo servidor...")
    http.ListenAndServe(":9000", nil)
}
```

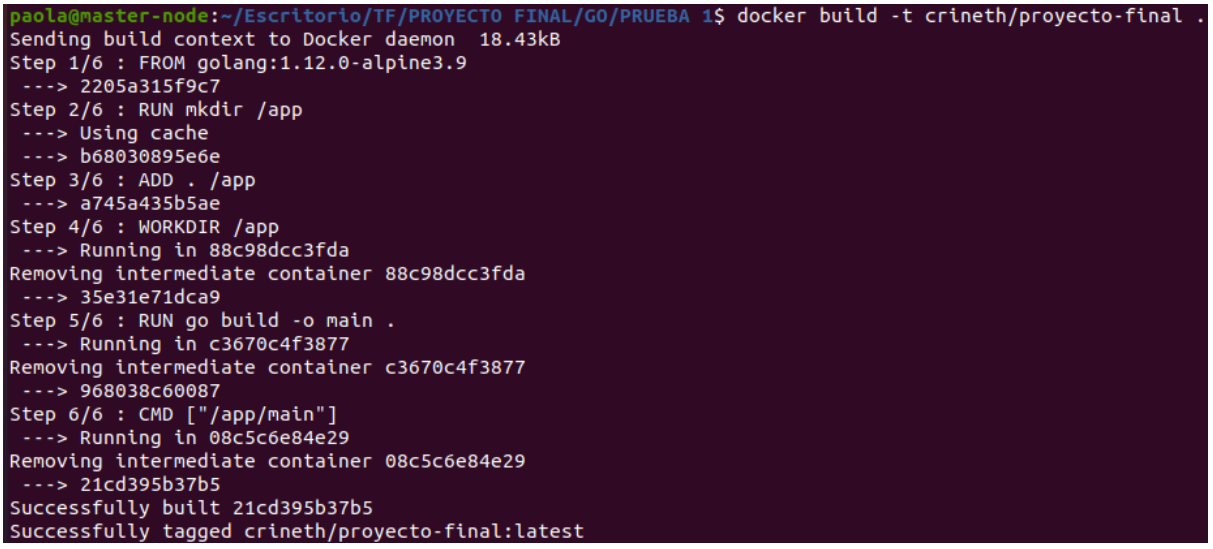
Para hacer posible la creación de la aplicación descentralizada se utilizó Docker Compose. Con dicha herramienta, se crearon varios nodos para simular el acceso desde diferentes pestañas a un mismo servidor web asegurándonos que todos estos pudieran realizar las transacciones en el programa (ya fuera de consulta o agregamiento) sin necesidad de hacer la conexión con otro servidor donde se estuviera almacenando la información, ya que esto la convertiría en una aplicación centralizada.
Con la utilización de Docker Compose, fue posible la creación, compilación y

ejecución del servicio, mismo que se puede conectar a muchos nodos pudiendo dar como resultado un sistema descentralizado o distribuido. En nuestro caso es descentralizado.
Además, es importante mencionar que Docker es una herramienta muy sencilla de usar que funciona a partir de contenedores virtualizando la ejecución de diversos comandos por medio de un archivo de configuración llamado Dockerfile. Esto ayuda al consumo de pocos recursos de la máquina donde se corra originalmente el proceso y, además, hacerlo de manera eficiente.
A continuación se muestra el código utilizado para hacer posible la ejecución del sistema por medio de Docker:

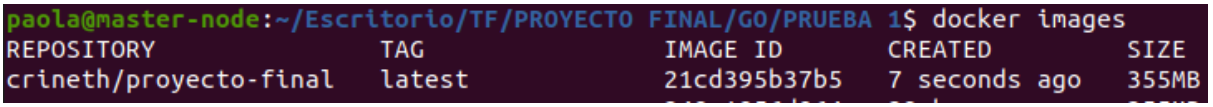
Para comenzar, se creó un archivo de configuración de Docker o Dockerfile, el cual incluía la ejecución del main del servidor previamente explicado:




Posteriormente, con el comando **docker build -t crineth/proyecto-final .** se compilaron dichos comandos obteniendo como resultado una imagen de Docker.




Para corroborar que la compilación hubiera sido exitosa, corrimos el comando **docker images** . A continuación se muestra el resultado:



Como es posible observar, la imagen estaba ahí, lista para ejecutarse.
Es importante mencionar que omitimos el paso de hacer el push del Dockerfile a la cuenta de Dockerhub debido a que previamente ya lo habíamos realizado.

Después corrimos el comando **docker run -it -p 80:8080 crineth/proyecto-final** con el fin de ejecutar el servicio en el puerto 80 y haciendo la comunicación con aplicaciones externas por medio del puerto 8080. Al correr el comando, marcó un error originalmente el cual indicaba que en el puerto 80 se estaba ejecutando otro proceso.

```
paola@master-node:~/Escritorio/TF/PROYECTO FINAL/GO/PRUEBA 1$ docker run -it -p 80:8080 crineth/proyecto-final
docker: Error response from daemon: driver failed programming external connectivity on endpoint kind_archimedes (eb5a1129ebbe41055fded3e537aae8d
502dda6f2a2b85c0cbd6ce505ff239602): Error starting userland proxy: listen tcp4 0.0.0.0:80: bind: address already in use.
ERRO[0000] error waiting for container: context canceled
```

Se determinó que la máquina era utilizada para pruebas de testing en un sistema que tenía nginx por lo que el siguiente paso fue correr el comando **sudo docker service nginx stop.** Al consultar el status fue posible observar que estaba apagado:

```
paola@master-node:~/Escritorio/TF/PROYECTO FINAL/GO/PRUEBA 1$ sudo service nginx status
● nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
     Active: inactive (dead) since Sun 2021-07-04 23:25:50 CDT; 8s ago
       Docs: man:nginx(8)
    Process: 925 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
    Process: 1013 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
    Process: 51372 ExecStop=/sbin/start-stop-daemon --quiet --stop --retry QUIT/5 --pidfile /run/nginx.pid (code=exited, status=0/SUCCESS)
   Main PID: 1037 (code=exited, status=0/SUCCESS)
```

Una vez realizado el movimiento fue posible mostrar en pantalla el formulario para agregar o consultar información:

```
paola@master-node:~/Escritorio/TF/PROYECTO FINAL/GO/PRUEBA 1$ docker run -it -p 80:8080 crineth/proyecto-final
Arrancando el servidor...
```

**Servidor Alumnos**

**Agregar alumno:**

Nombre del alumno:

Nombre de la materia:

Calificacion:

Agregar | Borrar informacion

**Promedio Alumno**

Nombre del alumno:

Consultar

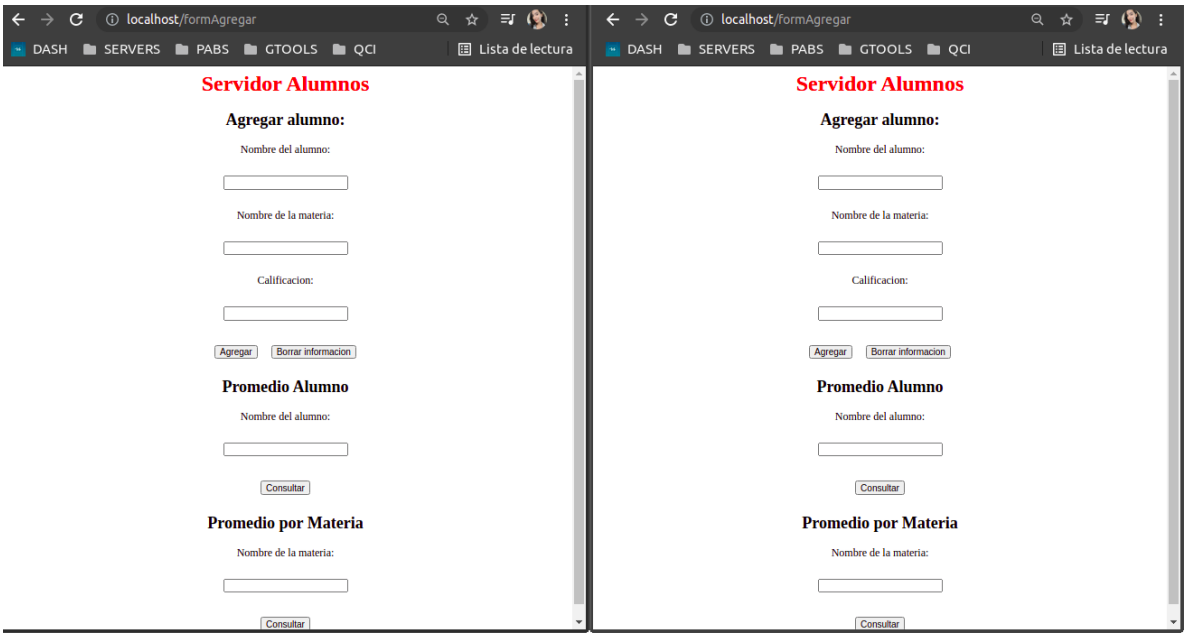
**Promedio por Materia**

Nombre de la materia:

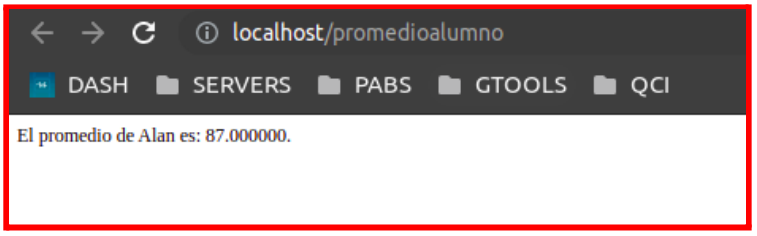
Consultar

## EJECUCIÓN

Debido a que el objetivo era realizar una aplicación descentralizada, una vez que el servidor web por medio de golang y la herramienta de Docker estuvieron sincronizadas y conectadas, abrimos dos pestañas conectadas al mismo puerto con el fin de mostrar “la magia de la aplicación”:

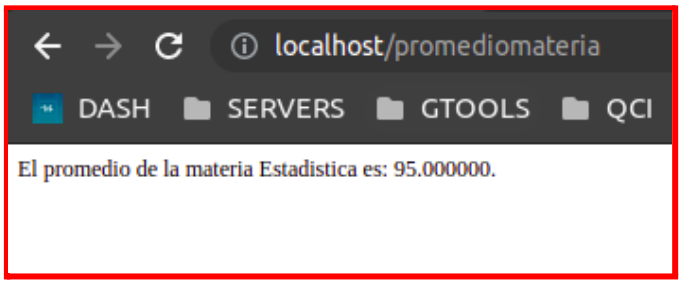



Posteriormente, en el nodo1 (navegador del lado izquierdo) agregamos un alumno llamado Alan con una calificación de 87 para la materia de Cálculo y unos segundos después consultamos desde el nodo2 (navegador del lado derecho) el promedio del alumno llamado Alan:




Después realizamos otra prueba desde el nodo 1 para consultar el promedio general de la materia de Estadística, teniendo como resultado que no existía dicha materia aún:

Así que desde el nodo dos procedemos a agregar 2 estudiantes: Maria con promedio de 100 y Saul con promedio de 90, ambos para la materia de estadística. Un segundo después de haberlo realizado, desde el nodo 1 volvemos a consultar el promedio para la materia de Estadística, teniendo como resultado lo siguiente:




## CONCLUSIÓN

Como conclusión podemos decir que las aplicaciones descentralizadas tienen sus pros y sus contras, en este caso específico nos dimos cuenta de que es muy útil usarlas cuando se quieren realizar transacciones sin necesidad de concentrar todo en un solo servidor, aunque para dicho fin es mucho más útil un sistema distribuido.

En la actualidad, tener un sistema descentralizado o distribuido implica un gran avance en la seguridad del sistema en general, debido a que las transacciones que se realizan son conocidas por todos y en caso de que se vea vulnerada la seguridad de algún nodo, otro podría contener la misma información sin necesidad de poner en riesgo la información y simplemente dando de baja al nodo afectado.

Consideramos que fue de gran ayuda ver este tema, investigar sobre ello y ver tutoriales para implementar nuestro sistema así como utilizar las herramientas que ya explicamos anteriormente