

Gestor de colas en parques temáticos

“Fun with Queues”



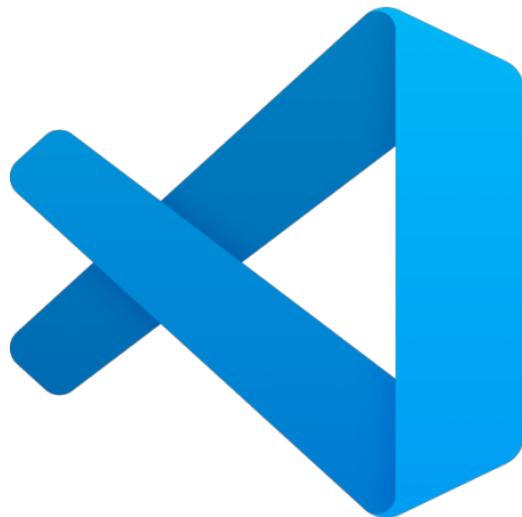
Jose Hurtado Baeza - 48824963B
Wilmer Fabricio Bravo Shuira - 29577276N
Sistemas Distribuidos
Práctica no guiada: Sockets, Eventos, Colas y modularidad
Curso 2021-2022

Entorno de desarrollo	3
Informe de los componentes software desarrollados	6
parque_atracciones(Base de Datos)	6
fwq_registry	9
fwq_sensor	15
fwq_waitingTimeServer	19
fwq_visitor	25
fwq_engine	42
Guía de despliegue de la aplicación	56
Capturas de pantalla que muestran el funcionamiento de las distintas aplicaciones conectadas	57

Entorno de desarrollo

Visual Studio Code

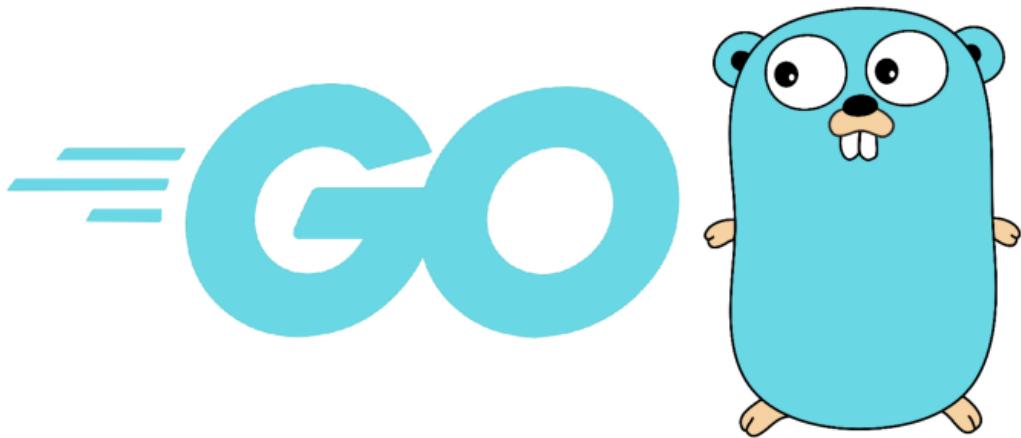
Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y web de código abierto para ofrecer a los usuarios una herramienta de programación avanzada. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias.



Lenguaje de programación golang

Go es un lenguaje de programación concurrente y compilado inspirado en la sintaxis de C, que intenta ser dinámico como Python y con el rendimiento de C o C++. Este lenguaje ha sido desarrollado por Google.

Go es un lenguaje de programación compilado, concurrente, imperativo, estructurado, orientado a objetos y con recolector de basura que de momentos es soportado por diferentes tipos de sistemas operativos. Las arquitecturas soportadas son i386, amd64 y ARM.



Apache Kafka

Apache Kafka es una plataforma de transmisión de eventos distribuida de código abierto utilizada por miles de empresas para canalizaciones de datos de alto rendimiento, análisis de transmisión, integración de datos y aplicaciones de misión crítica.



MySQL

Es un sistema de gestión de bases de datos relacional



MySQL Workbench

Es una herramienta visual para desarrolladores, administradores de bases de datos, etc. MySQL Workbench proporciona modelado de datos, desarrollo SQL y herramientas de administración integrales para la configuración del servidor, administración de usuarios, respaldo y mucho más.



Informe de los componentes software desarrollados

parque_atracciones(Base de Datos)

Este es el nombre de la base de datos relacional que hemos implementado en el entorno MySQL Workbench. Esta sirve para almacenar los recursos compartidos entre los distintos microservicios que forman el sistema central del parque.

Dicha base de datos fue creada mediante la interfaz de MySQL WorkBench y tras esto creamos un script llamado ParqueAtracciones.sql que es el que utilizamos para crear y llenar las tablas que nos hacían falta.

Vamos a comentar las 3 sentencias SQL de creación de tablas que son las realmente importantes, ya que las inserciones se realizaron de una forma más o menos “aleatoria”.

```
/* Creación de tablas */
CREATE TABLE parque (id varchar(30) PRIMARY KEY, aforoMaximo int, aforoActual int);
```

La primera sentencia es para la creación de la sencilla tabla “parque” en la que almacenamos en la columna “id” el identificador del parque, en la columna “aforoMaximo” el máximo número de visitantes que pueden estar en el parque en un mismo momento y en la columna “aforoActual” el número de visitantes que se encuentra en el parque actualmente.

```
CREATE TABLE visitante (
    id varchar(20) PRIMARY KEY,
    nombre varchar(30) NOT NULL,
    contraseña varchar(30) NOT NULL,
    posicionx int DEFAULT 0,
    posiciony int DEFAULT 0,
    destinox int DEFAULT -1,
    destinoy int DEFAULT -1,
    dentroParque int DEFAULT 0,
    idParque char(1),
    parqueAtracciones varchar(30) default 'SDpark',
    CONSTRAINT fk_visitantes_parque FOREIGN KEY (parqueAtracciones) REFERENCES parque (id));
```

La segunda sentencia crea la tabla “visitante” que nos va a servir para almacenar la información de los visitantes que entran al parque de atracciones.

En dicha tabla generamos la columna “id” para almacenar el identificador de cada visitante e indicamos que esta columna es clave primaria, ya que no nos interesa que haya 2 o más visitantes con el mismo identificador, sino no podríamos distinguirlos.

Después generamos las columnas “nombre” y “contraseña” para ya tener por completo las credenciales de acceso. Estas las declaramos como NOT NULL, ya que un visitante tiene que tener un nombre y un password asociado obligatoriamente para poder entrar al parque.

A continuación generamos las columnas “posicionx” y “posiciony” que nos indicarán en todo momento en qué lugar del parque se encuentra el visitante. A estas columnas les hemos añadido el valor 0 por defecto, ya que en nuestra implementación hemos decidido que la posición 0,0 sea la entrada al parque de atracciones.

Seguidamente tenemos las columnas “destinox” y “destinoy” las cuales hemos declarado con valor por defecto -1, ya que inicialmente el visitante no sabe a qué atracción va a querer dirigirse. Y por último, la columna parqueAtracciones que nos sirve para almacenar el parque de atracciones en el que se encuentra el visitante.

Luego tenemos la columna “dentroParque” que nos sirve para saber si el visitante ha iniciado sesión y se encuentra dentro del parque, para lo cual le daremos 1 como valor y 0 en caso contrario.

Acabando con las columnas de esta tabla, tenemos la columna “idParque” que

Además declaramos una clave ajena para hacer referencia al parque en el que se encuentra el visitante.

```

CREATE TABLE atraccion(
    id varchar(30) PRIMARY KEY,
    tciclo int,
    nvisitantes int,
    posicionx int,
    posiciony int,
    tiempoEspera int,
    parqueAtracciones varchar(30),
    CONSTRAINT fk_atracciones_parque FOREIGN KEY (parqueAtracciones) REFERENCES parque (id));

```

La tercera y última sentencia de creación de tablas es para la tabla “atraccion” que nos va a servir para almacenar la información de las atracciones que vamos a tener en nuestro parque.

Entrando en detalle, tenemos una columna “id” para guardar el identificador de la atracción, esta columna va a ser la clave primaria de la tabla, ya que no tiene sentido que 2 o más atracciones tengan el mismo identificador, ya que no podríamos distinguirlas fácilmente.

A continuación, tenemos la columna “tciclo” en la que almacenamos el tiempo (en minutos) que tarda la atracción en completar un ciclo de su funcionamiento y la columna “nvisitantes” que nos indica cuántas personas o más bien visitantes pueden montarse en la atracción en cada ciclo.

Seguidamente tenemos las columnas “posicionx” y “posiciony” que nos permiten saber en qué lugar del parque se encuentra instalada la atracción. Luego la columna “tiempoEspera” en la que almacenaremos el tiempo de espera de la atracción calculado por el Engine en base a la información proporcionada por el servidor de tiempos de espera, y finalmente la columna “parqueAtracciones” que declaramos como clave ajena ya que hace referencia al parque en el que se encuentra la atracción.

Por otro lado, tenemos los insert para la tabla atracción y es que en nuestro caso tenemos 16 atracciones:

```

/* Inserciones en las tablas */
INSERT INTO parque (id, aforoActual) VALUES ('SDpark', 0);
INSERT INTO atraccion (id, tciclo, nvisitantes, posicionx, posiciony, tiempoEspera, parqueAtracciones) VALUES ("atraccion1", 5, 3, 10, 14, 45, "SDpark");
INSERT INTO atraccion (id, tciclo, nvisitantes, posicionx, posiciony, tiempoEspera, parqueAtracciones) VALUES ("atraccion2", 8, 9, 1, 4, 30, "SDpark");
INSERT INTO atraccion (id, tciclo, nvisitantes, posicionx, posiciony, tiempoEspera, parqueAtracciones) VALUES ("atraccion3", 7, 6, 6, 6, 15, "SDpark");
INSERT INTO atraccion (id, tciclo, nvisitantes, posicionx, posiciony, tiempoEspera, parqueAtracciones) VALUES ("atraccion4", 18, 9, 10, 19, 65, "SDpark");
INSERT INTO atraccion (id, tciclo, nvisitantes, posicionx, posiciony, tiempoEspera, parqueAtracciones) VALUES ("atraccion5", 4, 5, 9, 17, 10, "SDpark");
INSERT INTO atraccion (id, tciclo, nvisitantes, posicionx, posiciony, tiempoEspera, parqueAtracciones) VALUES ("atraccion6", 10, 6, 3, 18, 40, "SDpark");
INSERT INTO atraccion (id, tciclo, nvisitantes, posicionx, posiciony, tiempoEspera, parqueAtracciones) VALUES ("atraccion7", 11, 8, 9, 2, 80, "SDpark");
INSERT INTO atraccion (id, tciclo, nvisitantes, posicionx, posiciony, tiempoEspera, parqueAtracciones) VALUES ("atraccion8", 19, 7, 2, 3, 90, "SDpark");
INSERT INTO atraccion (id, tciclo, nvisitantes, posicionx, posiciony, tiempoEspera, parqueAtracciones) VALUES ("atraccion9", 14, 6, 7, 8, 20, "SDpark");
INSERT INTO atraccion (id, tciclo, nvisitantes, posicionx, posiciony, tiempoEspera, parqueAtracciones) VALUES ("atraccion10", 15, 4, 18, 11, 13, "SDpark");
INSERT INTO atraccion (id, tciclo, nvisitantes, posicionx, posiciony, tiempoEspera, parqueAtracciones) VALUES ("atraccion11", 17, 7, 17, 5, 7, "SDpark");
INSERT INTO atraccion (id, tciclo, nvisitantes, posicionx, posiciony, tiempoEspera, parqueAtracciones) VALUES ("atraccion12", 7, 8, 6, 5, 17, "SDpark");
INSERT INTO atraccion (id, tciclo, nvisitantes, posicionx, posiciony, tiempoEspera, parqueAtracciones) VALUES ("atraccion13", 8, 2, 16, 17, 77, "SDpark");
INSERT INTO atraccion (id, tciclo, nvisitantes, posicionx, posiciony, tiempoEspera, parqueAtracciones) VALUES ("atraccion14", 12, 4, 19, 18, 40, "SDpark");
INSERT INTO atraccion (id, tciclo, nvisitantes, posicionx, posiciony, tiempoEspera, parqueAtracciones) VALUES ("atraccion15", 13, 10, 4, 15, 74, "SDpark");
INSERT INTO atraccion (id, tciclo, nvisitantes, posicionx, posiciony, tiempoEspera, parqueAtracciones) VALUES ("atraccion16", 19, 11, 15, 15, 23, "SDpark");

```

fwq_registry

El primer microservicio que vamos a comentar es el Registry, este se encargará del registro de los visitantes en el parque.

```
package main

import (
    "bufio"
    "database/sql"
    "fmt"
    "log"
    "net"
    "os"
    "strconv"
    "strings"
)

/* 
 * Estructura del visitante
 */
You, 3 weeks ago | 1 author (You)
type visitante struct {
    ID          string `json:"id"`
    Nombre      string `json:"nombre"`
    Password    string `json:"contraseña"`
    Posicionx   int    `json:"posicionx"`
    Posiciony   int    `json:"posiciony"`
    Destinox    int    `json:"destinox"`
    Destinoy    int    `json:"destinoy"`
    DentroParque int   `json:"dentroParque"`
    IdParque    string `json:"idParque"`
    Parque      string `json:"parqueAtracciones"`
}
```

En esta primera captura del código podemos ver la declaración del paquete, la importación de librerías que hemos necesitado y la creación del struct del visitante.

```

func main() {
    host := os.Args[1]
    puerto := os.Args[2]
    maxVisitantes, _ := strconv.Atoi(os.Args[3])

    // Arrancamos el servidor y atendemos conexiones entrantes
    fmt.Println("Arrancando el Registry, atendiendo en " + host + ":" + puerto)

    l, err := net.Listen("tcp", host+":"+puerto)

    if err != nil {
        fmt.Println("Error escuchando", err.Error())
        os.Exit(1)
    }

    // Cerramos el listener cuando se cierra la aplicación
    defer l.Close()

    // Bucle infinito hasta la salida del programa
    for {

        // Atendemos conexiones entrantes
        c, err := l.Accept()
        if err != nil {
            fmt.Println("Error conectando con un visitante: ", err.Error())
        }

        // Imprimimos la dirección de conexión del cliente
        log.Println("Visitante " + c.RemoteAddr().String() + " conectado.")

        // Llamamos a la función de forma asíncrona y manejamos las conexiones de forma concurrente
        go manejoConexion(c, maxVisitantes)
    }
}

```

En esta captura tenemos el bloque de código del main, donde primero guardamos la información pasada por parámetro, concretamente la ip y el puerto de escucha del Registry y el número máximo de visitantes que deberá coincidir con el pasado por parámetro al Engine.

A continuación, tenemos la parte donde arrancamos el registrador para que escuche por la ip y puerto indicados por parámetro, controlando el error en caso de fallo al ejecutar el net.Listen. Y además asegurándonos de que el listener se cerrará cuando se acabe el programa, esto lo hacemos con la declaración defer.

Después de esto, nos encontramos con un bucle infinito, ya que nos interesa que el Registry esté activo siempre para recibir peticiones por parte de los visitantes, para lo que tenemos la función Accept(). Al igual que antes, controlamos en caso de error imprimiendo un mensaje por pantalla relacionado y en caso de que no se produzca dicho error, imprimimos por pantalla la dirección de conexión del visitante que se ha conectado.

Finalmente llamamos de forma asíncrona a la función manejoConexion que trabajará de forma concurrente para poder atender a todos los visitantes que se conecten al Registry. En la llamada pasamos la conexión establecida vía socket y el máximo de visitantes que nos hará falta para controlar si el aforo del parque está completo o no.

```

/* Función que procesa concurrentemente los registros o actualizaciones de los visitantes */
func manejoConexion(conexion net.Conn) {
    // Lectura de la opción elegida por el visitante
    opcion, err := bufio.NewReader(conexion).ReadBytes('\n')

    // Cerramos la conexión de los clientes que se han desconectado
    if err != nil {
        fmt.Println("Visitante " + conexion.RemoteAddr().String() + " desconectado.")
        conexion.Close()
        return
    }

    You, a month ago • Creación y edición de perfiles ajustada, ahora el...
    opcionElegida := strings.TrimSpace(string(opcion))

    // Lectura del id del visitante hasta final de línea
    id, err := bufio.NewReader(conexion).ReadBytes('\n')

    // Cerramos la conexión de los clientes que se han desconectado
    if err != nil {
        fmt.Println("Visitante " + conexion.RemoteAddr().String() + " desconectado.")
        conexion.Close()
        return
    }

    // Lectura del nombre del visitante hasta final de línea
    nombre, err := bufio.NewReader(conexion).ReadBytes('\n')

    // Cerramos la conexión de los clientes que se han desconectado
    if err != nil {
        fmt.Println("Visitante " + conexion.RemoteAddr().String() + " desconectado.")
        conexion.Close()
        return
    }

    // Lectura del password del visitante hasta final de línea
    password, err := bufio.NewReader(conexion).ReadBytes('\n')

    // Cerramos la conexión de los clientes que se han desconectado
    if err != nil {
        fmt.Println("Visitante " + conexion.RemoteAddr().String() + " desconectado.")
        conexion.Close()
        return
    }
}

```

Entrando en el código de la función, aquí hacemos 4 lecturas a través de la conexión vía socket establecida con el Visitante, leyendo la opción elegida, el id, el nombre y el password del visitante que va a ser registrado o va a ser modificado, controlando en cada una de las lecturas si el visitante se ha desconectado para mostrar el mensaje de alerta.

```

// Si se ha solicitado un registro de usuario
if opcionElegida == "1" {

    // Imprimimos la información del visitante a registrar
    fmt.Println("Visitante a registrar -> ID: " + strings.TrimSpace(string(id)) +
    " | Nombre: " + strings.TrimSpace(string(nombre)) + " | Password: " + strings.TrimSpace(string(password)))

    // Si se ha solicitado editar/actualizar un perfil de usuario existente
} else if opcionElegida == "2" {

    // Imprimimos la información del visitante a editar
    fmt.Println("Visitante a editar -> ID: " + strings.TrimSpace(string(id)) +
    " | Nombre: " + strings.TrimSpace(string(nombre)) + " | Password: " + strings.TrimSpace(string(password)))

} else {
    conexion.Write([]byte("La opción elegida no es válida"))
    conexion.Close()
}

// Accedemos a la base de datos, empezando por abrir la conexión
db, err := sql.Open("mysql", "root:1234@tcp(127.0.0.1:3306)/parque_atracciones")

// Comprobamos que no haya error al conectarse
if err != nil {
    panic("Error al conectarse con la BD: " + err.Error())
}

defer db.Close() // Para que siempre se cierre la conexión con la BD al finalizar el programa

v := visitante{
    ID:      strings.TrimSpace(string(id)),
    Nombre:  strings.TrimSpace(string(nombre)),
    Password: strings.TrimSpace(string(password)),
    IdParque: strings.TrimSpace(string(id[0])),
}

```

Siguiendo con el código de la función, dependiendo de si se ha solicitado un registro o una modificación sacamos por pantalla la información del visitante de forma personalizada, y en caso contrario mostramos un error y cerramos la conexión. Si todo va bien abrimos la conexión a la base de datos, controlando con un panic para cerrar el programa en caso de que no se pueda acceder a la BD, añadiendo a continuación un defer para asegurarnos de que cuando finalice el programa se cierre siempre dicha conexión a la base de datos. A continuación creamos una instancia del struct visitante que nos va a servir para tratar la información del visitante.

```

// Si se ha solicitado un registro
if opcionElegida == "1" {

    results, err := db.Query("SELECT * FROM visitante WHERE id = ?", v.ID) // Comprueba si el visitante está registrado en la aplicación

    // Comprobamos que no se produzcan errores al hacer la consulta
    if err != nil {
        panic("Error al hacer la consulta de la información del visitante indicado: " + err.Error())
    }

    defer results.Close() // Nos aseguramos de cerrar

    // Si el visitante ya se había registrado
    if results.Next() {
        conexion.Write([]byte("El visitante ya está registrado en la aplicación"))
        conexion.Close()
    } else { // Si es un nuevo usuario

        results, err := db.Query("SELECT * FROM visitante") // Devuelve los visitantes que hay registrados en la aplicación

        // Comprobamos que no se produzcan errores al hacer la consulta
        if err != nil {
            panic("Error al hacer la consulta a la BD: " + err.Error())
        }

        defer results.Close() // Nos aseguramos de cerrar

        // Nos guardamos el nº actual de visitantes registrados en la aplicación
        visitantesActuales := 0 // You, 3 days ago • Ahora se controla en el Registry cuando se intent...
        for results.Next() {
            visitantesActuales += 1
        }

        // INSERTAMOS el nuevo visitante en la BD
        // Preparamos para prevenir inyecciones SQL
        sentenciaPreparada, err := db.Prepare("INSERT INTO visitante (id, nombre, contraseña, idParque) VALUES(?, ?, ?, ?)")
        if err != nil {
            panic("Error al preparar la sentencia de inserción: " + err.Error())
        }

        defer sentenciaPreparada.Close()

        // Ejecutar sentencia, un valor por cada '?'
        _, err = sentenciaPreparada.Exec(v.ID, v.Nombre, v.Password, v.IdParque)
        if err != nil {
            panic("Error al registrar el visitante: " + err.Error())
        }

        conexion.Write([]byte("Visitante registrado en el parque. Actualmente hay " + strconv.Itoa(visitantesActuales+1) + " visitantes registrados."))
        conexion.Close()
    }
}

```

Entonces, si se ha solicitado un registro, lo primero que hacemos es una consulta a la base de datos pidiendo que nos devuelva las filas de la tabla visitante donde el id se corresponda con el recibido por el visitante vía socket. Controlamos el error al hacer la consulta y nos aseguramos de cerrar la consulta a la base de datos con otro defer. Y gracias a esta consulta podemos comprobar si el visitante ya se encuentra registrado, en cuyo caso cerramos la conexión, no sin antes devolver un error a la aplicación visitante indicando el problema mencionado.

Por el contrario, si no se encuentra registrado, entonces procederemos a calcular el número de visitantes registrados para después insertar en la base de datos al nuevo visitante y así enviar un mensaje a la aplicación visitante en la que se informe que el visitante se ha registrado correctamente y además indicando el número total de visitantes registrados actualmente, finalizando con el cierre de la conexión.

```

} else { // Si se ha solicitado una actualización

    results, err := db.Query("SELECT * FROM visitante WHERE id = ?", v.ID) // Comprueba si el visitante está registrado en la aplicación

    // Comprobamos que no se produzcan errores al hacer la consulta
    if err != nil {
        panic("Error al hacer la consulta de la información del visitante indicado: " + err.Error())
    }

    defer results.Close() // Nos aseguramos de cerrar

    // Si el ID del visitante indicado por el cliente existe
    if results.Next() {

        // MODIFICAMOS la información de dicho visitante en la BD
        // Preparamos para prevenir inyecciones SQL
        sentenciaPreparada, err := db.Prepare("UPDATE visitante SET nombre = ?, contraseña = ? WHERE id = ?")
        if err != nil {
            panic("Error al preparar la sentencia de actualización: " + err.Error())
        }

        defer sentenciaPreparada.Close()

        // Ejecutar sentencia, un valor por cada '?'
        _, err = sentenciaPreparada.Exec(v.Nombre, v.Password, v.ID)
        if err != nil {
            panic("Error al modificar el visitante: " + err.Error())
        }

        conexion.Write([]byte("Visitante actualizado correctamente"))
        conexion.Close()

    } else {
        conexion.Write([]byte("El id del visitante no existe"))
        conexion.Close()
    }
}

// Reiniciamos el proceso
manejoConexion[conexion] You, a month ago • Creación y edición de perfiles ajustada, ahora el...
}

```

Por otra parte, si la petición es una modificación de la información de un visitante empezamos comprobando que el visitante se haya registrado previamente, después si es así, preparamos la sentencia a ejecutar, de tal manera que prevenimos inyecciones SQL, controlando en caso de error que se finalice el programa y se cierre la nueva conexión a la base de datos, y en caso de que todo haya ido bien, se ejecutará la sentencia que hemos preparado sustituyendo las interrogaciones con los valores indicados en la llamada a Exec, donde al igual que antes controlamos que no se produzca un error, y si todo se ejecuta correctamente envíamos vía socket un mensaje de respuesta al visitante informando de que su información ha sido actualizada, para finalmente cerrar la conexión. Si por el contrario, el visitante indicado en la petición no está registrado, enviamos un mensaje informando del error y cerramos la conexión sin hacer nada más.

Lo último que hacemos en el código de la función manejoConexion es precisamente hacer una llamada recursiva, de tal forma que el Registry se mantendrá en todo momento a la espera de peticiones por parte del visitante y manejando las conexiones con los visitantes de forma concurrente.

fwq_sensor

Vamos ahora con los sensores, estos son los dispositivos que se encargarán de enviar cada cierto tiempo aleatorio, el número de personas que hay en la cola de la atracción en la que dicho sensor está ubicado al servidor de tiempos de espera.

```
package main

import (
    "context"
    "fmt"
    "math/rand"
    "os"
    "strconv"
    "time"

    "github.com/segmentio/kafka-go"
)

You, 5 days ago | 1 author (You)
type sensor struct {
    IdAtraccion string
    Personas    int
}
```

Empezando a comentar el código, aquí podemos ver la declaración del paquete, las importaciones de librerías necesarias y la creación del struct `sensor` para facilitar el almacenamiento y el manejo de la información de un sensor.

```

func main() {
    ipBrokerGestorColas := os.Args[1]
    puertoBrokerGestorColas := os.Args[2]
    idAtraccion := os.Args[3] // Convertimos a entero
    crearTopic(ipBrokerGestorColas, puertoBrokerGestorColas)

    // Comprobamos que el id de la atracción sea válido
    valido := false
    for i := 1; !valido && i < 17; i++ {
        if idAtraccion == "atraccion"+strconv.Itoa(i) {
            valido = true
        }
    }

    // Si el id pasado por parámetro no es válido
    if !valido {
        panic("Error: El id de la atracción no es válido. Introduzca atraccion1...16")
    }

    brokerAddress := ipBrokerGestorColas + ":" + puertoBrokerGestorColas

    // Creamos un sensor
    s := new(sensor)
    s.IdAtraccion = idAtraccion

    // Generamos un número aleatorio de personas inicialmente en la cola de la atracción
    rand.Seed(time.Now().UnixNano()) // Utilizamos la función Seed(semilla) para inicializar la fuente predeterminada al requerir un comportamiento determinista
    min := 0
    max := 60
    s.Personas = (rand.Intn(max-min+1) + min)
    fmt.Println("Sensor creado para la atracción (" + idAtraccion + ") en la que inicialmente hay " + strconv.Itoa(s.Personas) + " personas en cola")

    // Generamos un tiempo aleatorio entre 1 y 3 segundos
    rand.Seed(time.Now().UnixNano()) // Utilizamos la función Seed(semilla) para inicializar la fuente predeterminada al requerir un comportamiento determinista
    min = 1
    max = 3
    tiempoAleatorio := (rand.Intn(max-min+1) + min)

    // Enviamos al servidor de tiempos el número de personas que se encuentra en la cola de la atracción
    enviaInformacion(s, brokerAddress, tiempoAleatorio)

    defer func() {
        fmt.Println("El sensor ha sido apagado.")
    }()
}

```

Respecto al main, en este primero nos guardamos los valores de los 3 parámetros introducidos al ejecutar el microservicio que son: La ip del broker gestor de colas, el puerto del broker gestor de colas y el identificador de la atracción en la base de datos.

A continuación, creamos el topic con una función auxiliar que aquí no vamos a comentar, a la cual le pasamos la ip y el puerto del broker gestor de colas, y dicha función nos va a generar de forma automática el topic llamado “sensor-servidorTiempos”.

Luego comprobamos que dicho id de la atracción introducido por parámetro sea válido, ya que si no se corresponde se generará un panic que mostrará la información del error adecuadamente por pantalla.

Después formamos la brokerAddress que le pasaremos a la función “envialinformacion” para poder conectarnos al gestor de colas que es el que mandará la información de los sensores al servidor de tiempos de espera.

Creamos a continuación un “objeto” sensor al que le asignamos la atracción pasada por parámetro. Seguidamente vamos a generar un par de números aleatorios, el primero para el

número de personas en la cola de la atracción y el segundo para el tiempo aleatorio que pasa entre cada envío de información por parte del sensor.

Y finalmente llamamos a la función “enviaInformacion” pasando el objeto sensor, la broker address y el tiempo aleatorio que pasa entre cada envío de información al servidor de tiempos de espera. Además con una función asíncrona notificamos al servidor de tiempos la caída del sensor.

```
/* Función que envía mediante un productor de Kafka la información recogida por el sensor */
func enviaInformacion(s *sensor, brokerAddress string, tiempoAleatorio int) {
    // Inicializamos el escritor
    escritor := kafka.NewWriter(kafka.WriterConfig{
        Brokers: []string{brokerAddress},
        Topic:   "sensor-servidorTiempos",
    })
    You, 2 months ago * Modificada la función enviaInformacion que ya no ...
    c := make(chan os.Signal, 1)
    signal.Notify(c, os.Interrupt)
    go func() {
        for sig := range c {
            log.Printf("captured %v, stopping profiler and exiting..", sig)
            mensaje := "\nSensor de la atracción (" + s.IdAtraccion + ") desconectado\n"
            err := escritor.WriteMessages(context.Background(),
                kafka.Message{
                    Key:  []byte("Atraccion " + s.IdAtraccion),
                    Value: []byte(mensaje),
                })
            if err != nil {
                panic("Error al conectarse al gestor de colas - No se puede mandar la información al servidor de tiempos de espera: " + err.Error())
            }
            fmt.Println()
            fmt.Println("Sensor desconectado manualmente")
            pprof.StopCPUProfile()
            os.Exit(1)
        }
    }()
}

for {
    err := escritor.WriteMessages(context.Background(),
        kafka.Message{
            Key:  []byte("Atraccion " + s.IdAtraccion),
            Value: []byte(s.IdAtraccion + ":" + strconv.Itoa(s.Personas)),
        })
    if err != nil {
        panic("Error al conectarse al gestor de colas - No se puede mandar la información al servidor de tiempos de espera: " + err.Error())
    }

    // Generamos un número aleatorio de personas en cola
    rand.Seed(time.Now().UnixNano()) // Utilizamos la función Seed(semilla) para inicializar la fuente predeterminada al requerir un comportamiento reproducible
    min := 0
    max := 60
    s.Personas = (rand.Intn(max-min+1) + min)

    // Cada 1 a 3 segundos el sensor envía la información al servidor de tiempos
    time.Sleep(time.Duration(tiempoAleatorio) * time.Second)

    fmt.Println("En la atracción [" + s.IdAtraccion + "] hay " + strconv.Itoa(s.Personas) + " personas en cola")
}
```

Continuando con la función, en esta vamos a generar un productor de Kafka que manda la información a la dirección indicada en la directiva “Brokers” a través del Topic “sensor-servidorTiempos”.

Para entonces abrir un bucle for infinito que continuamente mandará la información del sensor al servidor de tiempos. En este bucle por medio del productor de Kafka que hemos

generado indicamos la Key y el Value que envía un array de bytes donde ponemos el id de la atracción y el número de personas en cola de dicha atracción.

Controlamos además por si se produce un error al escribir el mensaje por el Kafka, que se nos muestre un mensaje por consola de información al respecto.

Y luego generamos de nuevo el número de personas en cola para que el servidor de tiempos de espera vaya actualizando sus tiempos de espera en base a las colas actuales.

Finalmente hacemos un time.Sleep con el tiempo aleatorio que generamos previamente para que el bucle se vuelva a ejecutar pasado dicho tiempo.

Para terminar con el microservicio tenemos la función de creación topics:

```
/*
 * Función que crea el topic para el envío de los movimientos de los visitantes
 */
func crearTopic(IPBroker, PuertoBroker string) {

    topic := "sensor-servidorTiempos"
    conn, err := kafka.Dial("tcp", IPBroker+":"+PuertoBroker)
    if err != nil {
        panic(err.Error())
    }
    defer conn.Close()

    controller, err := conn.Controller()

    if err != nil {
        panic(err.Error())
    }
    //Creamos una variable del tipo kafka.Conn
    var controllerConn *kafka.Conn
    //Le damos los valores necesarios para crear el controllerConn
    controllerConn, err = kafka.Dial("tcp", net.JoinHostPort(controller.Host, strconv.Itoa(controller.Port)))
    if err != nil {
        panic(err.Error())
    }
    defer controllerConn.Close()
    //Configuración del topic mapa-visitantes
    topicConfigs := []kafka.TopicConfig{
        kafka.TopicConfig{      redundant type from array, slice, or map composite literal
            Topic:          topic,
            NumPartitions: 1,
            ReplicationFactor: 1,
        },
    }
    err = controllerConn.CreateTopics(topicConfigs...)
    if err != nil {
        panic(err.Error())
    }
}
```

En esta empezamos indicando el topic a crear, preparando el dial y asegurándonos de cerrar dicha conexión.

Luego creamos y configuramos el controller que utilizamos a posteriori para la creación del topic que hemos indicado, al cual le podemos personalizar el número de particiones y el valor de replication factor.

fwq_waitingTimeServer

Continuamos con el servidor de tiempos de espera, el cual implementa la funcionalidad necesaria para conocer el tiempo de espera de cada atracción. Este microservicio recibe a través del gestor de colas, por parte de los sensores, el número de visitantes que hay en las colas de cada atracción. Y además, permanece a la escucha indefinidamente esperando a que el Engine le pida los tiempos de espera de todas las atracciones.

```
package main

import (
    "bufio"
    "context"
    "database/sql"
    "fmt"
    "net"
    "os"
    "strconv"
    "strings"

    _ "github.com/go-sql-driver/mysql"
    _ "github.com/segmentio/kafka-go"
)

/*
 * Estructura de las atracciones
 */

You, 6 days ago | 1 author (You)
type atraccion struct {
    ID          string `json:"id"`
    TCiclo      int    `json:"tciclo"`
    NVisitantes int    `json:"nvisitantes"`
    Posicionx   int    `json:"posicionx"`
    Posiciony   int    `json:"posiciony"`
    TiempoEspera int   `json:"tiempoEspera"`
    Parque      string `json:"parqueAtracciones"`
}
```

Como siempre, declaramos el paquete, las librerías importadas y además de esto, generamos un struct “atraccion” para poder gestionar más fácilmente la información de las atracciones.

```

func main() {

    host := os.Args[1]
    puertoEscucha := os.Args[2]
    ipBrokerGestorColas := os.Args[3]
    puertoBrokerGestorColas := os.Args[4]

    crearTopic(ipBrokerGestorColas, puertoBrokerGestorColas)

    var conexionBD = conexionBD()
    var atracciones []atraccion

    atracciones, _ = obtenerAtraccionesBD(conexionBD)

    // Arrancamos el servidor y atendemos conexiones entrantes
    fmt.Println("Servidor de tiempos atendiendo en " + host + ":" + puertoEscucha)

    l, err := net.Listen("tcp", host+":"+puertoEscucha)

    if err != nil {
        fmt.Println("Error escuchando", err.Error())
        os.Exit(1)
    }

    // Cerramos el listener cuando se cierra la aplicación
    defer l.Close()

    // Bucle infinito hasta la salida del programa
    for {

        // Atendemos conexiones entrantes
        c, err := l.Accept()
        if err != nil {
            fmt.Println("Error conectando con el engine:", err.Error())
        }

        // Imprimimos la dirección de conexión del cliente
        log.Println("Cliente engine " + c.RemoteAddr().String() + " conectado.")

        // Manejamos las conexiones de forma concurrente
        go manejoConexion(ipBrokerGestorColas, puertoBrokerGestorColas, c, atracciones)

    }
}

```

En el main empezamos guardando la información pasada por parámetro donde tenemos la ip y puerto de escucha del servidor de tiempos y la ip y el puerto del broker gestor de colas.

A continuación generamos el topic a utilizar para la recepción de la información por parte de los sensores.

La función utilizada para ello es la siguiente:

```
/*
 * Función que crea el topic para el envio de los movimientos de los visitantes
 */
func crearTopic(IpBroker, PuertoBroker string) {

    topic := "sensor-servidorTiempos"
    conn, err := kafka.Dial("tcp", IpBroker+":"+PuertoBroker)
    if err != nil {
        panic(err.Error())
    }
    defer conn.Close()

    controller, err := conn.Controller()

    if err != nil {
        panic(err.Error())
    }
    //Creamos una variable del tipo kafka.Conn
    var controllerConn *kafka.Conn
    //Le damos los valores necesarios para crear el controllerConn
    controllerConn, err = kafka.Dial("tcp", net.JoinHostPort(controller.Host, strconv.Itoa(controller.Port)))
    if err != nil {
        panic(err.Error())
    }
    defer controllerConn.Close()
    //Configuración del topic mapa-visitantes
    topicConfigs := []kafka.TopicConfig{
        kafka.TopicConfig{      redundant type from array, slice, or map composite literal
            Topic:          topic,
            NumPartitions: 1,
            ReplicationFactor: 1,
        },
    }
    err = controllerConn.CreateTopics(topicConfigs...)
    if err != nil {
        panic(err.Error())
    }
}
```

Luego hacemos una llamada a la función auxiliar conexionBD que nos abre una conexión a la base de datos.

Esta es la función mencionada:

```
/*
 * Función que abre una conexión con la bd
 */
func conexionBD() *sql.DB {
    //Accediendo a la base de datos
    //Abrimos la conexión con la base de datos
    db, err := sql.Open("mysql", "root:1234@tcp(127.0.0.1:3306)/parque_atracciones")
    //Si la conexión falla mostrara este error
    if err != nil {
        panic(err.Error())
    }
    //Cierra la conexión con la bd
    //defer db.Close()
    return db
}
```

Y nos creamos una array de atracciones para gestionar, almacenar o modificar la información de todas las atracciones que hay en el parque o también en la base de datos.

Dichas atracciones las obtenemos en la sentencia siguiente con la llamada a la función auxiliar “obtenerAtraccionesBD”.

Esta es la función mencionada:

```
/*
 * Función que obtiene las atracciones del parque
 * @return []atraccion : Array con las atracciones del parque
 * @return error : Error en caso de que no se ha podido obtener las atracciones
 */
func obtenerAtraccionesBD(db *sql.DB) ([]atraccion, error) {

    //Ejecutamos la sentencia
    results, err := db.Query("SELECT * FROM atraccion")
    if err != nil {
        return nil, err //devolverá nil y error en caso de que no se pueda hacer la consulta
    }

    //Cerramos la base de datos
    defer results.Close()

    //Declaramos el array de visitantes
    var atraccionesParque []atraccion

    //Recorremos los resultados obtenidos por la consulta
    for results.Next() {
        //  var nombreVariable tipoVariable
        //Variable donde guardamos la información de cada una filas de la sentencia
        var fwq_atraccion atraccion
        if err := results.Scan(&fwq_atraccion.ID, &fwq_atraccion.TCiclo,
            &fwq_atraccion.NVisitantes, &fwq_atraccion.Posicionx,
            &fwq_atraccion.Posiciony, &fwq_atraccion.TiempoEspera,
            &fwq_atraccion.Parque); err != nil {
            return atraccionesParque, err
        }
        //Vamos añadiendo las atracciones al array
        atraccionesParque = append(atraccionesParque, fwq_atraccion)
    }

    if err = results.Err(); err != nil {
        return atraccionesParque, err
    }
    return atraccionesParque, nil
}
```

Y ya entrando en lo importante, hacemos una llamada asíncrona y concurrente a la función manejoConexion que es la función que se va a encargar de recibir la información de los sensores por medio de un consumidor Kafka. Dicha llamada está englobada en un bucle for en el aceptamos las conexiones entrantes, es decir, las peticiones del engine para que le enviemos los tiempos de espera de las atracciones actualizados.

```
// Función que maneja la lógica para una única petición de conexión
func manejoConexion(IpBroker, PuertoBroker string, conn net.Conn, atracciones []atencion) {

    // Lectura del buffer de entrada hasta el final de línea
    buffer, err := bufio.NewReader(conn).ReadBytes('\n')

    fmt.Println("Petición del Engine: " + string(buffer))

    // Cerrar las conexiones con engines desconectados
    if err != nil {
        log.Println("Engine" + conn.RemoteAddr().String() + " desconectado.")
        conn.Close()
        return
    }

    broker := IpBroker + ":" + PuertoBroker
    r := kafka.ReaderConfig(kafka.ReaderConfig{
        Brokers:      []string{broker},
        Topic:        "sensor-servidorTiempos",
        StartOffset:  kafka.LastOffset,
    })

    reader := kafka.NewReader(r)
}
```

En esta función tenemos un consumidor Kafka que lee a través del broker y el topic indicados y a partir del último registro, cosa que indicamos mediante el uso de la propiedad StartOffset puesta a kafka.LastOffset.

```

for {

    m, err := reader.ReadMessage(context.Background())
    if err != nil {
        fmt.Println("Ha ocurrido algún error a la hora de conectarse con kafka", err)
    }

    if strings.Contains(string(m.Value), "desconectado") {
        fmt.Println(string(m.Value))
        conn.Close()
    } else {
        fmt.Println("[", string(m.Value)+" personas en cola", "]")

        infoSensor := strings.Split(string(m.Value), ":")

        idAtraccion := infoSensor[0]
        personasEnCola, _ := strconv.Atoi(infoSensor[1])

        encontrado := false

        // Buscamos la atracción indicada por el sensor para calcular su tiempo de espera actual
        for i := 0; i < len(atracciones) && !encontrado; i++ {

            if atracciones[i].ID == idAtraccion {
                encontrado = true
                atracciones[i].TiempoEspera = calculaTiempoEspera(atracciones[i], personasEnCola)
            }
        }

        tiemposEspera := ""

        // Formamos la cadena con los tiempos de espera que le vamos a mandar al engine
        for i := 0; i < len(atracciones); i++ {
            tiemposEspera += atracciones[i].ID + ":" + strconv.Itoa(atracciones[i].TiempoEspera) + "|"
        }

        // Mandamos una cadena separada por barras con los tiempos de espera de cada atracción al engine
        conn.Write([]byte(tiemposEspera))
        conn.Close()
    }
}
}

```

Creamos un bucle for para que se atienda de forma concurrente a los sensores y comprobamos que no se haya producido un error al conectarse al gestor de colas.

En caso de que el mensaje recibido sea una notificación de un sensor que se ha desconectado lo tratamos debidamente.

En caso contrario, nos guardamos la información pasada por los sensores y luego buscamos la atracción indicada por el sensor para calcular su tiempo de espera actual por medio de la función “calculaTiempoEspera”:

```
/* Función que calcula el tiempo de espera de una atracción dada */
func calculaTiempoEspera(a atraccion, personasEnCola int) int {

    tiempoEspera := 0

    // Mientras queden personas en la cola de la atracción
    for personasEnCola > a.NVisitantes {

        tiempoEspera += a.TCiclo
        personasEnCola -= a.NVisitantes

    }

    return tiempoEspera
}
```

Esta función se basa en el número de personas en cola y el número de visitantes que pueden montarse en una atracción en un único ciclo, por lo que mientras las personas en cola sean mayores a las personas que pueden subir a la atracción, vamos a ir incrementando el tiempo de espera sumando el tiempo de ciclo en cada iteración.

Luego formamos la cadena con los tiempos de espera actualizados que le mandamos al engine vía socket y finalmente cerramos la conexión.

fwq_visitor

Los visitantes son las personas que entrarán al parque de atracciones, primero tendrán que registrarse. En este caso solo tienen que introducir su ID, nombre y contraseña.

Después tendrán una opción de edición de perfil donde podrán cambiar alguna información. Después tendrá la opción de moverse por el parque de atracciones, el cual se irá moviendo a las atracciones que tengan un tiempo de espera menor de 60 minutos y finalmente podrán salir del parque de atracciones.

Los visitantes cuentan con tres tópicos. Uno por el cual irán enviando las peticiones al engine, otro por el que leen la respuesta del engine a la petición de login y otro por donde recibirán el mapa por parte del engine.

Ahora comentaré algunas características principales del código.

```
package main

import (
    "bufio"
    "context"
    "crypto/rand"
    "encoding/json"
    "fmt"
    "log"
    "math"
    "math/rand"
    "net"
    "os"
    "os/signal"
    "runtime/pprof"
    "strconv"
    "strings"
    "time"

    "github.com/oklog/ulid/v2"
    "github.com/segmentio/kafka-go"
)
```

Primero empezamos definiendo la paqueterías y librerías necesarias.

```

/*
 * Estructura del visitante
 */
You, 3 weeks ago | 1 author (You)
type visitante struct {
    ID      string `json:"id"`
    Nombre  string `json:"nombre"`
    Password string `json:"contraseña"`
    Posicionx int   `json:"posicionx"`
    Posiciony int   `json:"posiciony"`
    Destinox int   `json:"destinox"`
    Destinoy int   `json:"destinoy"`
    DentroParque int `json:"dentroParque"`
    IdParque  string `json:"idParque"`
    Parque    string `json:"parqueAtracciones"`
}

/*
 * Estructura de las atracciones
 */
You, 2 weeks ago | 1 author (You)
type atraccion struct { You, 2 weeks ago • Im
    ID      string `json:"id"`
    TCiclo  int   `json:"tciclo"`
    NVisitantes int `json:"nvisitantes"`
    Posicionx int   `json:"posicionx"`
    Posiciony int   `json:"posiciony"`
    TiempoEspera int `json:"tiempoEspera"`
    Parque   string `json:"parqueAtracciones"`
}

const (
    connType = "tcp"
)

```

Luego definimos los structs visitante y atraccion junto con la constante connType que indica el tipo de conexión que va a utilizar el socket para conectarnos al registry.

```

var v = visitante{ // Guardamos la información del visitante que nos hace falta
    ID:      "",
    Password:      "",
    Posicionx:    0,
    Posiciony:    0,
    Destinox:    -1,
    Destinoy:    -1,
    DentroParque: 0,
}

var a = atraccion{ // Guardamos la información de la atraccion que nos hace falta
    Posicionx:    -1,
    Posiciony:    -1,
    TiempoEspera: -1,
}

```

Ahora nos declaramos un par de variables globales para gestionar la información del visitante y de la atracción destino de dicho visitante.

```

/**
 * Función main de los visitantes
 */
func main() {

    //Argumentos iniciales
    IpFWQ_Registry := os.Args[1]
    PuertoFWQ := os.Args[2]
    IpBroker := os.Args[3]
    PuertoBroker := os.Args[4]
    crearTopic(IpBroker, PuertoBroker, "peticiones")
    crearTopic(IpBroker, PuertoBroker, "respuesta-login")
    crearTopic(IpBroker, PuertoBroker, "movimiento-mapa")
    fmt.Println("/**Bienvenido al parque de atracciones**")
    fmt.Println()
    MenuParque(IpFWQ_Registry, PuertoFWQ, IpBroker, PuertoBroker)

}

```

Este es el main de la aplicación visitante, en el primero capturamos la información pasada por parámetro, luego generamos los 3 topics necesarios y por último llamamos a la función MenuParque que nos va a mostrar el menú de la aplicación.

En la siguiente imagen veremos la función que implementa la creación de un visitante.

```
/* Función que se conecta al módulo FWQ_Registry para crear un nuevo usuario */
func CrearPerfil(ipRegistry, puertoRegistry string) {

    fmt.Println("*****Creación de perfil*****")
    conn, err := net.Dial(connType, ipRegistry+":"+puertoRegistry)

    if err != nil {
        fmt.Println("Error al conectarse al Registry:", err.Error())
    } else { // Si el visitante establece conexión con el Registry indicado por parámetro

        conn.Write([]byte("1" + "\n")) // Le pasamos al Registry la opción elegida por el visitante

        reader := bufio.NewReader(os.Stdin)

        fmt.Print("Introduce tu ID:")
        id, _ := reader.ReadString('\n')
        conn.Write([]byte(id))

        fmt.Print("Introduce tu nombre:")
        nombre, _ := reader.ReadString('\n')
        conn.Write([]byte(nombre))

        fmt.Print("Introduce tu contraseña:")
        password, _ := reader.ReadString('\n')
        conn.Write([]byte(password))

        //Escuchando por el relay el mensaje de respuesta del Registry
        message, _ := bufio.NewReader(conn).ReadString('\n')

        // Comprobamos si el Registry nos devuelve un mensaje de respuesta
        if message != "" {
            log.Println("Respuesta del Registry: ", message)
        } else {
            log.Println("Lo siento, el Registry no está disponible en estos momentos.")
        }
    }
}
```

Donde le pasamos como argumentos de la función la ip y el puerto del registry.

En la función se pueden dar dos situaciones, la primera es en la que el Registry no está disponible por lo que se produce un error al tratar de establecer una conexión vía socket y la segunda es en la que si nos conectamos y envíamos el id, el nombre y la contraseña del visitante, tras lo cual escuchamos la respuesta del registry a la petición para verificar si se ha resuelto satisfactoriamente o si por el contrario, el registry ha caído durante el proceso.

Ahora veremos la edición de perfil que es muy similar al registro.

```
/* Función que se conecta al módulo FWQ_Registry para editar o actualizar el perfil de un usuario existente */
func EditarPerfil(ipRegistry, puertoRegistry string) {

    fmt.Println("Has entrado a editar perfil")
    conn, err := net.Dial(connType, ipRegistry+":"+puertoRegistry)

    if err != nil {
        fmt.Println("Error al conectarse al Registry:", err.Error())
    } else { // Si el visitante establece conexión con el Registry indicado por parámetro

        conn.Write([]byte("2" + "\n")) // Le pasamos al Registry la opción elegida por el visitante

        reader := bufio.NewReader(os.Stdin)

        fmt.Println("Información del visitante que se quiere modificar:")
        fmt.Print("Introduce el ID:")
        id, _ := reader.ReadString('\n')
        conn.Write([]byte(id))

        fmt.Print("Introduce el nombre:")
        nombre, _ := reader.ReadString('\n')
        conn.Write([]byte(nombre))

        fmt.Print("Introduce la contraseña:")
        password, _ := reader.ReadString('\n')
        conn.Write([]byte(password))

        message, _ := bufio.NewReader(conn).ReadString('\n')

        // Comprobamos si el Registry nos devuelve un mensaje de respuesta
        if message != "" {
            log.Println("Respuesta del Registry: ", message)
        } else {
            log.Println("Lo siento, el Registry no está disponible en estos momentos.")
        }
    }
}
```

Como podemos ver le pasamos como parámetros la ip y el puerto del registry al igual que en el caso de la función de registro.

Y realmente el código es muy similar al caso anterior, sólo que ahora envíamos la opción 2 en lugar de la opción 1 para que el Registry sepa que lo que queremos es un update de nuestros datos.

Con relación a la entrada al parque por parte de los visitantes tenemos el siguiente código.

```
/* Función que envía las credenciales de acceso del visitante para entrar en el parque */
func EntradaParque(ipRegistry, puertoRegistry, IpBroker, PuertoBroker string) {
    fmt.Println("*Bienvenido al parque de atracciones*")
    reader := bufio.NewReader(os.Stdin)
    fmt.Print("Por favor introduce tu alias:")
    alias, _ := reader.ReadString('\n')
    v.ID += strings.TrimSpace(string(alias))

    fmt.Print("y tu password:")
    password, _ := reader.ReadString('\n')
    v.Password += strings.TrimSpace(string(password))

    ctx := context.Background()
    mensaje := strings.TrimSpace(string(alias)) + ":" + strings.TrimSpace(string(password)) + ":" + strconv.Itoa(v.Destinox) + "," + strconv.Itoa(v.Destinoy)

    // Mandamos al engine las credenciales de inicio de sesión del visitante para entrar al parque
    productorLogin(IpBroker, PuertoBroker, mensaje, ctx)

    c := make(chan os.Signal, 1)
    signal.Notify(c, os.Interrupt)
    go func() {
        for sig := range c {
            log.Printf("captured %v, stopping profiler and exiting..", sig)
            mensaje := v.ID + ":" + "OUT" + ":" + strconv.Itoa(v.Destinox) + "," + strconv.Itoa(v.Destinoy)
            productorSalir(IpBroker, PuertoBroker, mensaje, ctx)
            fmt.Println()
            fmt.Println("Adios, esperamos que haya disfrutado su estancia en el parque.")
            pprof.StopCPUProfile()
            os.Exit(1)
        }
    }()
    // Recibe del engine el mapa actualizado o un mensaje de parque cerrado
    consumidorLogin(ipRegistry, puertoRegistry, IpBroker, PuertoBroker, ctx)
}
```

Donde el visitante introduce su id/alias así como su contraseña y mandará por un tópico la información proporcionada para solicitar la entrada al parque, es decir, el visitante envía una petición de inicio de sesión y para ello tenemos implementada la función productorLogin:

```
/* Función que se encarga de enviar las credenciales de inicio de sesión */
func productorLogin(IpBroker, PuertoBroker, credenciales string, ctx context.Context) {
    var brokerAddress string = IpBroker + ":" + PuertoBroker
    var topic string = "peticiones"

    w := kafka.NewWriter(kafka.WriterConfig{
        Brokers:          []string{brokerAddress},
        Topic:           topic,
        CompressionCodec: kafka.Snappy.Codec(),
    })

    err := w.WriteMessages(ctx, kafka.Message{
        Key:   []byte("Key-Login"),
        Value: []byte(credenciales),
    })
    if err != nil {
        panic("No se pueden encolar las credenciales: " + err.Error())
    }

    fmt.Println("Enviando credenciales -> " + credenciales)
}
```

Siguiendo con la función de entrada al parque, a continuación del envío de la petición de login tenemos declarada una función anónima y asíncrona que sirve para que en el momento en el que un visitante se quiera desconectar de la aplicación se envíe una petición de salida utilizando para ello la función llamada productorSalir:

```
/* Función que se encarga de enviar las credenciales de inicio de sesión */
func productorLogin(IpBroker, PuertoBroker, credenciales string, ctx context.Context) {

    var brokerAddress string = IpBroker + ":" + PuertoBroker
    var topic string = "peticiones"

    w := kafka.NewWriter(kafka.WriterConfig{
        Brokers:          []string{brokerAddress},
        Topic:            topic,
        CompressionCodec: kafka.Snappy.Codec(),
    })

    err := w.WriteMessages(ctx, kafka.Message{
        Key:   []byte("Key-Login"),
        Value: []byte(credenciales),
    })
    if err != nil {
        panic("No se pueden encolar las credenciales: " + err.Error())
    }

    fmt.Println("Enviando credenciales -> " + credenciales)
}
```

Volviendo a la función de entrada al parque, dos aspectos a destacar es la creación del topic en cuanto ponemos en marcha el visitante y la forma de consumir y producir por parte de los visitantes. Esta función termina llamando a la función consumidorMapa:

```
/* Función que recibe el mapa del engine y lo devuelve formateado */
func consumidorMapa(IpBroker, PuertoBroker string, ctx context.Context) {
    broker := IpBroker + ":" + PuertoBroker
    r := kafka.ReaderConfig(kafka.ReaderConfig{
        Brokers: []string{broker},
        Topic:   "movimiento-mapa",
        GroupID: Ulid(),
        //De esta forma solo cogera los ultimos mensajes despues de unirse al cluster
        StartOffset: kafka.LastOffset,
    })
    reader := kafka.NewReader(r)
}
```

El consumidor lo configuramos de la siguiente forma. Donde el nombre del topic es mapa-visitantes, este es el canal por donde recibirá el mapa de parte del engine. Cabe recalcar que tenemos la opción de StartOffset, la cual hace que cada visitante que se una al tópico empiece a leer desde el último mensaje que ha llegado al topic. Pero un detalle muy relevante respecto a la configuración del consumidor es que utilizamos un GroupID aleatorio, y esto lo hacemos para que cada nuevo proceso de la aplicación visitante posea un GroupID distinto, lo cual nos ha ayudado a resolver conflictos a la hora de tener varios visitantes escuchando de forma simultánea por el mismo engine.

Y para la generación de dichos groupId aleatorios hemos utilizado la siguiente función:

```
func Ulid() string {
    t := time.Now().UTC()
    id := ulid.MustNew(ulid.Timestamp(t), hho.Reader)

    return id.String()
}
```

Por otro lado, respecto a cómo creamos los topics en golang.

Le pasamos por parámetros la ip y puerto por donde kafka va a estar escuchando, lo configuramos conforme nos decía la documentación oficial de kafka en go.

```

/*
 * Función que crea el topic para el envío de los movimientos de los visitantes
 */
func crearTopic(IpBroker, PuertoBroker, topic string) {

    var broker1 string = IpBroker + ":" + PuertoBroker
    conn, err := kafka.Dial("tcp", broker1)
    if err != nil {
        panic(err.Error())
    }
    defer conn.Close()

    controller, err := conn.Controller()

    if err != nil {
        panic(err.Error())
    }

    //Creamos una variable del tipo kafka.Conn
    var controllerConn *kafka.Conn
    //Le damos los valores necesarios para crear el controllerConn
    controllerConn, err = kafka.Dial("tcp", net.JoinHostPort(controller.Host, strconv.Itoa(controller.Port)))
    if err != nil {
        panic(err.Error())
    }
    defer controllerConn.Close()
    //Configuración del topic mapa-visitantes
    topicConfigs := []kafka.TopicConfig{
        kafka.TopicConfig{ Topic: topic,
                           NumPartitions: 10,
                           ReplicationFactor: 1,
        },
    }
    err = controllerConn.CreateTopics(topicConfigs...)
    if err != nil {
        panic(err.Error())
    }
}

```

Como podemos ver, algunos de los valores son similares a la forma en la que creamos los topics como si hiciéramos de forma manual.

```

for v.DentroParque == 1 {

    m, err := reader.ReadMessage(context.Background())

    if err != nil {
        panic("Ha ocurrido algún error a la hora de conectarse con kafka: " + err.Error())
    }
    var mapaObtenido string
    err = json.Unmarshal(m.Value, &mapaObtenido)

    if err != nil {
        fmt.Println("Error al decodificar el mapa: %v\n", err)      fmt.Println call has possible formatting directive %v
    }

    // Como el parque ha cerrado tenemos que reiniciar la información del visitante
    if mapaObtenido == "Engine no disponible" {
        fmt.Println("El engine ha dejado de estar disponible")
        v.DentroParque = 0
        v.ID = ""
        v.Password = ""
    } else {
        // Procesamos el mapa recibido y lo convertimos a un array bidimensional de strings
        cadenaProcesada := strings.Split(string(m.Value), "|")
        var mapa [20][20]string = procesarMapa(cadenaProcesada)
        fmt.Println(mapaObtenido)
        movimiento := obtenerMovimiento(mapa)
        peticionMovimiento := v.ID + ":" + movimiento + ":" + strconv.Itoa(v.Destinox) + "," + strconv.Itoa(v.Destinoy)
        productorMovimientos(IpBroker, PuertoBroker, peticionMovimiento, ctx)
        time.Sleep(1 * time.Second) // Mandamos el movimiento del visitante cada segundo
    }
}

```

Continuando con el código de la función consumidorMapa tenemos un bucle for que hará que se quede escuchando por ese tópico siempre y cuando el visitante se mantenga dentro del parque.

Dentro de dicho bucle hacemos la lectura de o bien el mapa o bien de un mensaje en formato json por parte del engine indicando que ha dejado de estar disponible, en cuyo caso pararemos las iteraciones del bucle indicando que el visitante ya no está en el parque y reiniciando la información de la variable global que gestiona al visitante actual por si en el próximo inicio de sesión se quiere utilizar otro visitante. Tras esto volvemos al menú de la aplicación.

En caso de que el mensaje recibido sea el mapa entonces procesamos la cadena recibida para formar el mapa con la función procesarMapa:

```

/* Función que formatea el mapa y lo convierte en un array bidimensional de strings */
func procesarMapa(mapa []string) [20][20]string {
    var mapaFormatado [20][20]string
    k := 0
    for i := 0; i < 20; i++ {
        for j := 0; j < 20; j++ {
            if k < 400 {
                mapaFormatado[i][j] = mapa[k]
                k++
            }
        }
    }
    return mapaFormatado
}

```

Siguiendo con el código de la función consumidorMapa, mostramos el mapa procesado, y ahora que tenemos el mapa disponible nos toca calcular/obtener el movimiento que va a realizar el visitante, para ello llamamos a la función obtenerMovimiento:

```

/* Función que se encarga de ir moviendo al visitante hasta alcanzar el destino */
func obtenerMovimiento(mapa [20][20]string) string {
    var movimiento string
    // Si el visitante no sabe a qué atracción dirigirse o la atracción actual elegida tiene un tiempo de espera mayor a 60 minutos
    if v.Destinox == -1 || v.Destinoy == -1 || a.TiempoEspera >= 60 {
        seleccionaAtraccionAlAzar(mapa)
    } else {
        actualizaAtraccion(mapa) // Actualizamos el tiempo de espera de la atracción destino del visitante
    }

    movimiento = calculaMovimiento() // Obtiene el mejor movimiento en base a las posiciones adyacentes y la atracción destino seleccionada
    actualizaPosicion(movimiento) // Actualiza la posición actual del visitante en base al mejor movimiento elegido

    // Si el visitante se encuentra en la atracción
    if (v.Posicionx == v.Destinox) && (v.Posiciony == v.Destinoy) {
        time.Sleep(10 * time.Second) // Esperamos un tiempo para simular el tiempo de ciclo de la atracción

        // Ahora el visitante vuelve a desconocer su destino
        v.Destinox = -1
        v.Destinoy = -1
        a.TiempoEspera = -1
        a.Posicionx = -1
        a.Posiciony = -1
    }
    return movimiento
}

```

Esta función actuará de forma diferente dependiendo de si el visitante desconoce a qué atracción dirigirse/su atracción actual ha cambiado su tiempo de espera y ahora es mayor o igual a 60 minutos o si por el contrario no se cumple ninguna de estas dos condiciones.

En el primer caso tenemos que seleccionar una atracción al azar y para ello disponemos de la función `seleccionaAtraccionAlAzar`:

```
/* Función que selecciona una atracción al azar y guarda la posición de dicha atracción en el visitante */
func seleccionaAtraccionAlAzar(mapa [20][20]string) {

    var atraccionesDisponibles []atraccion

    //Elegimos una atracción al azar del mapa entre las que el tiempo de espera sea menor de 60 minutos
    for i := 0; i < 20; i++ {
        for j := 0; j < 20; j++ {
            if t, err := strconv.Atoi(mapa[i][j]); err == nil { // Si la posición actual del mapa es un número
                if t < 60 { // Si el tiempo de espera es menor a 60 minutos
                    atraccionAux := atraccion{
                        Posicionx: i,
                        Posiciony: j,
                        TiempoEspera: t,
                    }
                    atraccionesDisponibles = append(atraccionesDisponibles, atraccionAux)
                }
            }
        }
    }

    // Elegimos al azar una de las atracciones disponibles
    rand.Seed(time.Now().UnixNano()) // Utilizamos la función Seed(semilla) para inicializar la fuente predeterminada
    min := 0
    max := len(atraccionesDisponibles) - 1
    indexAtraccion := (rand.Intn(max-min+1) + min)

    // Actualizamos las coordenadas de destino del visitante
    v.Destinox = atraccionesDisponibles[indexAtraccion].Posicionx
    v.Destinoy = atraccionesDisponibles[indexAtraccion].Posiciony
    a.Posicionx = atraccionesDisponibles[indexAtraccion].Posicionx
    a.Posiciony = atraccionesDisponibles[indexAtraccion].Posiciony
    a.TiempoEspera = atraccionesDisponibles[indexAtraccion].TiempoEspera

    fmt.Println("Me dirijo a la atracción con tiempo de espera igual a " + strconv.Itoa(a.TiempoEspera))
}
```

Esta se encarga de seleccionar al azar entre las atracciones del mapa que no posean un tiempo de espera mayor o igual a 60 minutos y tras lo cual almacena la información de las coordenadas de dicha atracción en la variable global que gestiona el visitante y en la variable global que hace referencia a la atracción destino del visitante, donde además guardamos el tiempo de espera.

Volviendo a la función obtenerMovimiento, en el caso en que no se cumple ninguna de las condiciones mencionadas anteriormente, y por lo tanto, significando que el visitante se dirige a una atracción lo que tenemos que hacer es actualizar el tiempo de espera de su atracción destino por si hubiese sido actualizado desde el engine tras una petición al servidor de tiempos de espera. Para esto disponemos de la función actualizaAtracción:

```
/* Función que actualiza el tiempo de espera de la atracción destino del visitante en base al mapa recibido */
func actualizaAtraccion(mapa [20][20]string) {
    a.TiempoEspera, _ = strconv.Atoi(mapa[a.Posicionx][a.Posiciony])
}
```

Siguiendo con el código de la función obtenerMovimiento, ahora calculamos el movimiento con la función calculaMovimiento:

```
/* Función que devuelve el mejor movimiento a realizar en base a la atracción destino elegida por el visitante */
func calculaMovimiento() string {
    var mejorMovimiento string = ""
    var mejorDistancia int
    var nuevaDistancia int

    xOriginal := v.Posicionx
    yOriginal := v.Posiciony

    // Seleccionamos el mejor movimiento para que el visitante alcance su destino
    for i := 0; i < 8; i++ {

        switch i {
        case 0:
            v.Posicionx--
            if v.Posicionx == -1 {
                v.Posicionx = 19
            }
            mejorDistancia = int(math.Abs(float64(v.Destinox)-float64(v.Posicionx))) + int(math.Abs(float64(v.Destinoy)-float64(v.Posiciony))) // Distancia de Manhattan
            mejorMovimiento = "N"
            v.Posicionx = xOriginal // Reseteamos la posición
        case 1:
            v.Posicionx++
            if v.Posicionx == 20 {
                v.Posicionx = 0
            }
            nuevaDistancia = int(math.Abs(float64(v.Destinox)-float64(v.Posicionx))) + int(math.Abs(float64(v.Destinoy)-float64(v.Posiciony))) // Distancia de Manhattan
            if nuevaDistancia < mejorDistancia {
                mejorDistancia = nuevaDistancia
                mejorMovimiento = "S"
            }
            v.Posicionx = xOriginal // Reseteamos la posición
        case 2:
            v.Posiciony--
            if v.Posiciony == -1 {
                v.Posiciony = 19
            }
            nuevaDistancia = int(math.Abs(float64(v.Destinox)-float64(v.Posicionx))) + int(math.Abs(float64(v.Destinoy)-float64(v.Posiciony))) // Distancia de Manhattan
            if nuevaDistancia < mejorDistancia {
                mejorDistancia = nuevaDistancia
                mejorMovimiento = "W"
            }
            v.Posiciony = yOriginal // Reseteamos la posición
        }
    }
    return mejorMovimiento
}
```

```

        case 3:
            v.Posiciony++
            if v.Posiciony == 20 {
                v.Posiciony = 0
            }
            nuevaDistancia = int(math.Abs(float64(v.Destinox)-float64(v.Posicionx)) + int(math.Abs(float64(v.Destinoy)-float64(v.Posiciony))) // Distancia de Manhattan
            if nuevaDistancia < mejorDistancia {
                mejorDistancia = nuevaDistancia
                mejorMovimiento = "E"
            }
            v.Posiciony = yOriginal // Reseteamos la posición
        case 4:
            v.Posicionx--
            v.Posiciony-
            if v.Posicionx == -1 {
                v.Posicionx = 19
            }
            if v.Posiciony == -1 {
                v.Posiciony = 19
            }
            nuevaDistancia = int(math.Abs(float64(v.Destinox)-float64(v.Posicionx)) + int(math.Abs(float64(v.Destinoy)-float64(v.Posiciony))) // Distancia de Manhattan
            if nuevaDistancia < mejorDistancia {
                mejorDistancia = nuevaDistancia
                mejorMovimiento = "NW"
            }
            v.Posicionx = xOriginal // Reseteamos la posición
            v.Posiciony = yOriginal // Reseteamos la posición
        case 5:
            v.Posicionx-
            v.Posiciony+
            if v.Posicionx == -1 {
                v.Posicionx = 19
            }
            if v.Posiciony == 20 {
                v.Posiciony = 0
            }
            nuevaDistancia = int(math.Abs(float64(v.Destinox)-float64(v.Posicionx)) + int(math.Abs(float64(v.Destinoy)-float64(v.Posiciony))) // Distancia de Manhattan
            if nuevaDistancia < mejorDistancia {
                mejorDistancia = nuevaDistancia
                mejorMovimiento = "NE"
            }
            v.Posicionx = xOriginal // Reseteamos la posición
            v.Posiciony = yOriginal // Reseteamos la posición
    }
}

```

```

case 6:
    v.Posicionx++
    v.Posiciony-
    if v.Posicionx == 20 {
        v.Posicionx = 0
    }
    if v.Posiciony == -1 {
        v.Posiciony = 19
    }
    nuevaDistancia = int(math.Abs(float64(v.Destinox)-float64(v.Posicionx)) + int(math.Abs(float64(v.Destinoy)-float64(v.Posiciony))) // Distancia de Manhattan
    if nuevaDistancia < mejorDistancia {
        mejorDistancia = nuevaDistancia
        mejorMovimiento = "SW"
    }
    v.Posicionx = xOriginal // Reseteamos la posición
    v.Posiciony = yOriginal // Reseteamos la posición
case 7:
    v.Posicionx++
    v.Posiciony+
    if v.Posicionx == 20 {
        v.Posicionx = 0
    }
    if v.Posiciony == 20 {
        v.Posiciony = 0
    }
    nuevaDistancia = int(math.Abs(float64(v.Destinox)-float64(v.Posicionx)) + int(math.Abs(float64(v.Destinoy)-float64(v.Posiciony))) // Distancia de Manhattan
    if nuevaDistancia < mejorDistancia {
        mejorDistancia = nuevaDistancia
        mejorMovimiento = "SE"
    }
    v.Posicionx = xOriginal // Reseteamos la posición
    v.Posiciony = yOriginal // Reseteamos la posición
}

return mejorMovimiento
}

```

Aquí básicamente lo que hacemos es probar las 8 posibles posiciones y calcular la distancia de Manhattan respecto a cada una de ellas y nos quedaremos con la posición que menor distancia tenga respecto a la atracción destino, y de esta forma devolveremos el mejor movimiento posible en consecuencia.

Volviendo a la función obtenerMovimiento lo siguiente es actualizar la posición del visitante en función del movimiento elegido, y para ello tenemos la función actualizaPosicion:

```
/* Función que actualiza la posición actual del visitante en base al movimiento pasado */
func actualizaPosicion(movimiento string) {

    switch movimiento {

        case "N":
            v.Posicionx--
        case "S":
            v.Posicionx++
        case "W":
            v.Posiciony--
        case "E":
            v.Posiciony++
        case "NW":
            v.Posicionx--
            v.Posiciony--
        case "NE":
            v.Posicionx--
            v.Posiciony++
        case "SW":
            v.Posicionx++
            v.Posiciony--
        case "SE":
            v.Posicionx++
            v.Posiciony++
    }

    if v.Posicionx == -1 {
        v.Posicionx = 19
    } else if v.Posicionx == 20 {
        v.Posicionx = 0
    }
}
```

En esta función dependiendo del movimiento elegido actuamos en consecuencia actualizando la posición actual del visitante.

```
// Si el visitante se encuentra en la atracción
if (v.Posicionx == v.Destinox) && (v.Posiciony == v.Destinoy) {

    time.Sleep(10 * time.Second) // Esperamos un tiempo para simular el tiempo de ciclo de la atracción

    // Ahora el visitante vuelve a desconocer su destino
    v.Destinox = -1
    v.Destinoy = -1
    a.TiempoEspera = -1
    a.Posicionx = -1
    a.Posiciony = -1

}

return movimiento
```

Y finalizando con la función obtenerMovimiento en caso de que se llegue a la atracción destino hacemos una espera de 10 segundos para simular el tiempo de ciclo de la atracción y reiniciamos los valores de la posición del visitante y de la atracción. Para terminar con la función devolvemos un string con el movimiento elegido que es el que envíamos al engine para que lo procese y nos actualice la posición del visitante actual en el mapa.

```

    peticionMovimiento := v.ID + ":" + movimiento + ":" + strconv.Itoa(v.Destinox) + "," + strconv.Itoa(v.Destinoy)
    productorMovimientos([IpBroker, PuertoBroker, peticionMovimiento, ctx]) You, 3 days ago • Controlada la caid
    time.Sleep(1 * time.Second) // Mandamos el movimiento del visitante cada segundo
}
}

```

Y volviendo a la función consumidorMapa sólo nos queda mandar la petición con el movimiento al engine para que nos actualice nuestra posición en el mapa, para ello utilizamos la función productorMovimientos:

```

/* Función que se encarga de enviar los movimientos de los visitantes al engine */
func productorMovimientos(IpBroker, PuertoBroker, movimiento string, ctx context.Context) {

    var brokerAddress string = IpBroker + ":" + PuertoBroker
    var topic string = "peticiones"

    w := kafka.NewWriter(kafka.WriterConfig{
        Brokers:          []string{brokerAddress},
        Topic:            topic,
        CompressionCodec: kafka.Snappy.Codec(),
    })

    err := w.WriteMessages(ctx, kafka.Message{
        Key:   []byte("Key-Moves"),
        Value: []byte(movimiento),
    })
    if err != nil {
        panic("No se puede encolar el movimiento: " + err.Error())
    }

    fmt.Println("Enviando movimiento: " + movimiento)
}

```

Para finalizar con la función consumidorMapa hacemos una espera de 1 segundo para enviar el siguiente movimiento como se nos dice en el enunciado.

fwq_engine

El engine es el sistema central de la aplicación, el cual pondrá en marcha todo el sistema. Utiliza tres tópicos, los cuales son los mismos que los que tiene el visitante. Uno para recibir la peticiones por parte de los visitantes, otro para enviar las respuestas de login y otro para enviar el mapa.

Lo primero que nos encontramos en el código del engine es la declaración de la paquetería y la importación de las librerías necesarias:

```
package main

import (
    "bufio"
    "context"
    "database/sql"
    "encoding/json"
    "fmt"
    "log"
    "net"
    "os"
    "os/signal"
    "runtime/pprof"
    "strconv"
    "strings"
    "time"

    _ "github.com/go-sql-driver/mysql"
    "github.com/segmentio/kafka-go"
```

Lo segundo que tenemos son las estructuras creadas, las cuales representan a los visitantes y a las atracciones:

```
/*
 * Estructura del visitante
 */
You, 3 weeks ago | 2 authors (You and others)
type visitante struct {
    ID      string `json:"id"`
    Nombre  string `json:"nombre"`
    Password string `json:"contraseña"`
    Posicionx int    `json:"posicionx"`
    Posiciony int    `json:"posiciony"`
    Destinox  int   `json:"destinox"`
    Destinoy  int   `json:"destinoy"`
    DentroParque int  `json:"dentroParque"`
    IdParque   string `json:"idParque"`
    Parque    string `json:"parqueAtracciones"`
}

/*
 * Estructura de las atracciones
 */
Wilmer Bravo, a month ago | 1 author (Wilmer Bravo)
type atraccion struct {
    ID      string `json:"id"`
    TCiclo  int    `json:"tciclo"`
    NVisitantes int   `json:"nvisitantes"`
    Posicionx int    `json:"posicionx"`
    Posiciony int    `json:"posiciony"`
    TiempoEspera int  `json:"tiempoEspera"`
    Parque   string `json:"parqueAtracciones"`
}
```

Ahora entramos en el main de la aplicación:

```
/*
 * @Description : Función main de fwq_engine
 * @Author : Wilmer Fabricio Bravo Shuira
 */
func main() {

    IpKafka := os.Args[1]
    PuertoKafka := os.Args[2]
    numeroVisitantes := os.Args[3]
    IpFWQWaiting := os.Args[4]
    PuertoWaiting := os.Args[5]

    //Creamos el topic...Cambiar la IpKafka en la función principal
    //Si no se ejecuta el programa, se cierra el kafka?
    crearTopics(IpKafka, PuertoKafka, "peticiones")
    crearTopics(IpKafka, PuertoKafka, "respuesta-login")
    crearTopics(IpKafka, PuertoKafka, "movimiento-mapa")

    // Visitantes, atracciones que se encuentran en la BD
    var visitantesRegistrados []visitante
    var conn = conexionBD()
    maxVisitantes, _ := strconv.Atoi(numeroVisitantes)
    establecerMaxVisitantes(conn, maxVisitantes)
    //Para empezar con el kafka
    ctx := context.Background()
    go consumidorEngine(IpKafka, PuertoKafka, ctx, maxVisitantes)
```

Primero nos guardamos los valores introducidos por parámetro, luego nos creamos los 3 topics necesarios, después nos preparamos las variables que van a almacenar a los visitantes registrados, la conexión a la base de datos y el número máximo de visitantes.

Para obtener la conexión a la base de datos hacemos uso de la siguiente función:

```
/*
 * Función que abre una conexión con la bd
 */
func conexionBD() *sql.DB {
    //Accediendo a la base de datos
    //****Flate blod ****
    //Abrimos la conexión con la base de datos
    db, err := sql.Open("mysql", "root:1234@tcp(127.0.0.1:3306)/parque_atracciones")
    //Si la conexión falla mostrara este error
    if err != nil {
        panic(err.Error())
    }
    //Cierra la conexión con la bd
    defer db.Close()
    return db
}
```

Y para establecer el máximo número de visitantes en la BD utilizamos esta:

```
/*
 * Función que establece el aforo maximo permitido en el parque de atracciones
 */
func establecerMaxVisitantes(db *sql.DB, numero int) {

    //Ejecutamos la sentencia
    results, err := db.Query("SELECT * FROM parque")

    if err != nil {
        panic("Error al hacer la consulta del parque" + err.Error()) //devolverá nil y error en caso de que no se pueda hacer la consulta
    }

    //Cerramos la base de datos
    defer results.Close()

    //Recorremos los resultados obtenidos por la consulta
    if results.Next() {

        //Variable donde guardamos la información de cada una filas de la sentencia
        sentenciaPreparada, err := db.Prepare("UPDATE parque SET aforoMaximo = ? WHERE id = ?")

        if err != nil {
            panic("Error al preparar la sentencia" + err.Error()) //devolverá nil y error en caso de que no se pueda hacer la consulta
        }

        defer sentenciaPreparada.Close()

        _, err = sentenciaPreparada.Exec(numero, "SDpark")

        if err != nil {
            panic("Error al establecer el número máximo de visitantes" + err.Error())
        }
    }
}
```

A continuación tenemos la llamada a la función asíncrona consumidorEngine que trabajará de forma asíncrona y concurrente para procesar las peticiones realizadas por parte los visitantes.

Entrando en dicha función, lo primero que nos encontramos en la definición/configuración del consumidor de kafka junto con una nueva apertura de conexión a la base de datos que no dependa de la que trabaja en el hilo de ejecución principal del main:

```
/* Función que recibe del gestor de colas las credenciales de los visitantes que quieren iniciar sesión para entrar en el parque */
func consumidorEngine(IpKafka, PuertoKafka string, ctx context.Context, maxVisitantes int) {

    //Accediendo a la base de datos
    //Abrimos la conexión con la base de datos
    db, err := sql.Open("mysql", "root:1234@tcp(127.0.0.1:3306)/parque_atracciones")
    //Si la conexión falla mostrara este error
    if err != nil {
        panic(err.Error())
    }
    //Cierra la conexión con la bd
    defer db.Close()

    direcciónKafka := IpKafka + ":" + PuertoKafka
    //Configuración de lector de kafka
    conf := kafka.ReaderConfig{
        Brokers:      []string{direcciónKafka},
        Topic:        "peticiones", //Topico que hemos creado
        GroupID:      "visitantes",
        StartOffset:  kafka.LastOffset,
    }

    reader := kafka.NewReader(conf)
}
```

Continuando con el código de la función aparece el bucle for donde iremos leyendo los mensajes que han sido encolados por los visitante en el gestor de colas:

```
for {

    m, err := reader.ReadMessage(context.Background())

    if err != nil {
        fmt.Println("Ha ocurrido algún error a la hora de conectarse con el kafka", err)
    }

    //fmt.Println("Petición recibida: " + string(m.Value))

    cadenaPeticion := strings.Split(string(m.Value), ":")

    alias := cadenaPeticion[0]
    petición := cadenaPeticion[1]
    destino := strings.Split(cadenaPeticion[2], ",")
    destinoX, _ := strconv.Atoi(strings.TrimSpace(destino[0]))
    destinoY, _ := strconv.Atoi(strings.TrimSpace(destino[1]))

    v := visitante{
        ID:      strings.TrimSpace(alias),
        Password: strings.TrimSpace(petición),
        Destinox: destinoX,
        Destinoy: destinoY,
    }

    // Comprobamos si lo enviado son credenciales de acceso en cuyo caso se trata de una petición de login
    results, err := db.Query("SELECT * FROM visitante WHERE id = ? and contraseña = ?", v.ID, v.Password)

    if err != nil {
        fmt.Println("Error al hacer la consulta sobre la BD para el login: " + err.Error())
    }

    var respuesta string = ""
```

En este bucle procesamos los mensajes recibidos y obtenemos el alias del visitante, la petición realizada, y las coordenadas de destino actuales de dicho visitante. A continuación nos creamos una instancia del struct visitante para gestionar/almacenar dicha información del visitante y tras esto hacemos una consulta a la base de datos que nos va a permitir comprobar si lo recibido son credenciales de acceso, en cuyo caso se trata de una petición de login para entrar en el parque. Para terminar con esta captura nos vamos ya preparando la variable respuesta donde almacenaremos la respuesta a la petición de login.

Ahora entramos en las diferentes casuísticas que se pueden presentar en esta función.

```
// Si las credenciales coinciden con las de un visitante registrado en la BD y el parque no está lleno
if results.Next() && !parqueLleno(db, maxVisitantes) {
    // Actualizamos el estado del visitante en la BD
    sentenciaPreparada, err := db.Prepare("UPDATE visitante SET dentroParque = 1, destinox = ?, destinoy = ? WHERE id = ?")
    if err != nil {
        panic("Error al preparar la sentencia de modificación: " + err.Error())
    }
    u, 2 weeks ago * Falta ver porque no recibe las atracciones actual...
    // Ejecutar sentencia, un valor por cada '?'
    _, err = sentenciaPreparada.Exec(v.Destinox, v.Destinoy, v.ID)
    if err != nil {
        panic("Error al actualizar el estado del visitante respecto al parque: " + err.Error())
    }

    respuesta += alias + ":" + "Acceso concedido"
    productorLogin(IpKafka, PuertoKafka, ctx, respuesta)
}
sentenciaPreparada.Close()
```

La primera de ellas se trata del caso en el que hemos recibido una petición de login, por lo que comprobamos si las credenciales coinciden con las de un visitante registrado en la BD y que el parque no esté lleno actualmente.

En caso de que dichas condiciones se cumplan, actualizamos el estado del visitante en el BD, estableciendo su atracción destino actual, la cual será desconocida (valores -1 y -1) e indicando que ahora se encuentra dentro del parque.

Si todo va bien formamos el mensaje de respuesta donde le indicamos al visitante que le permitimos el acceso al parque y lo envíamos utilizando para ello la función productorLogin:

```
/* Función que envía el mensaje de respuesta a la petición de login de un visitante */
func productorLogin(IpBroker, PuertoBroker string, ctx context.Context, respuesta string) {
    var brokerAddress string = IpBroker + ":" + PuertoBroker
    var topic string = "respuesta-login"

    w := kafka.NewWriter(kafka.WriterConfig{
        Brokers:          []string{brokerAddress},
        Topic:            topic,
        CompressionCodec: kafka.Snappy.Codec(),
    })

    err := w.WriteMessages(ctx, kafka.Message{
        Key:   []byte("Key-Login"),
        Value: []byte(respuesta),
    })

    if err != nil {
        fmt.Println("No se puede mandar el mensaje de respuesta a la petición de login: " + err.Error())
    }
}
```

La segunda de las casuísticas que se pueden dar es la referente a una solicitud de movimiento:

```
else if peticion == "IN" || peticion == "N" || peticion == "S" || peticion == "W" || peticion == "E" || peticion == "NW" ||
peticion == "NE" || peticion == "SW" || peticion == "SE" { // Si se nos ha mandado un movimiento

    // Comprobamos que el alias pertenezca a un visitante que se encuentra en el parque
    results, err := db.Query("SELECT * FROM visitante WHERE id = ?", v.ID)

    if err != nil {
        fmt.Println("Error al hacer la consulta sobre la BD para el login: " + err.Error())
    }

    // Actualizamos el estado el destino del visitante en la BD
    sentenciaPreparada, err := db.Prepare("UPDATE visitante SET destinox = ?, destinoy = ? WHERE id = ?")
    if err != nil {
        panic("Error al preparar la sentencia de modificación de destino: " + err.Error())
    }

    // Ejecutar sentencia, un valor por cada '?'
    _, err = sentenciaPreparada.Exec(v.Destinox, v.Destinoy, v.ID)
    if err != nil {
        panic("Error al actualizar el destino del visitante: " + err.Error())
    }

    sentenciaPreparada.Close()
```

En este caso tenemos varios posibles movimientos a recibir, pero si hay uno que quiero destacar es el “IN” que hace referencia al movimiento ejercido por el visitante para indicarnos que ha entrado en el parque. El resto de movimientos son los que se mencionan en el enunciado y que se comportan tal y como indican sus siglas.

Entrando en la lógica de este bloque de código, primero hacemos una select para luego comprobar que el alias indicado en la petición de movimiento pertenezca a un visitante que se encuentra en el parque y luego actualizamos el destino del visitante en la BD.

```
if results.Next() {

    var mapa [20][20]string
    visitantesParque, _ := obtenerVisitantesParque(db)           // Obtenemos los visitantes del parque actualizados
    mueveVisitante(db, alias, peticion, visitantesParque)        // Movemos al visitante en base al movimiento recibido
    visitantesParqueActualizados, _ := obtenerVisitantesParque(db) // Obtenemos los visitantes del parque actualizados
    // Preparamos el mapa a enviar a los visitantes que se encuentra en el parque
    atracciones, _ := obtenerAtraccionesBD(db) // Obtenemos las atracciones actualizadas
    mapaActualizado := asignacionPosiciones(visitantesParqueActualizados, atracciones, mapa)
    var representacion string
    for i := 0; i < len(mapaActualizado); i++ {
        for j := 0; j < len(mapaActualizado); j++ {
            if j == 19 {
                representacion = representacion + mapaActualizado[i][j] + "\n"
            } else {
                representacion = representacion + mapaActualizado[i][j]
            }
        }
    }
    //Convertimos el mapaActualizado a formato json
    //Esta función devuelve un array de byte
    mapaJson, err := json.Marshal(representacion)
    //En formato json tiene cuenta el salto de linea por lo que hay que ver si al decodificarlo se quita
    if err != nil {
        fmt.Println("Error a la hora de codificar el mapa: %v", err) // fmt.Println call has possible formatting directive %v
    }
    productorMapa(IpKafka, PuertoKafka, ctx, mapaJson) // Mandamos el mapa actualizado a los visitantes que se encuentran en el parque
    results.Close()
```

En caso que el id de la petición sea válido vamos a formar el mapa.

Para ello empezamos obteniendo los visitantes que se encuentran actualmente en el parque, utilizando para ello la función obtenerVisitantesParque:

```
/*
 * Función que obtiene todos los visitantes que se encuentran en el parque de la BD
 * @return []visitante : Arrays de los visitantes obtenidos en la sentencia
 * @return error : Error en caso de que no se haya podido obtener ninguno
 */
func obtenerVisitantesParque(db *sql.DB) ([]visitante, error) {

    //Ejecutamos la sentencia
    results, err := db.Query("SELECT * FROM visitante WHERE dentroParque = 1")

    if err != nil {
        return nil, err //devolvera nil y error en caso de que no se pueda hacer la consulta
    }

    //Cerramos la base de datos
    defer results.Close()

    //Declaramos el array de visitantes
    var visitantes []visitante

    //Recorremos los resultados obtenidos por la consulta
    for results.Next() {

        //Variable donde guardamos la información de cada una filas de la sentencia
        var fwq_visitante visitante

        if err := results.Scan(&fwq_visitante.ID, &fwq_visitante.Nombre,
            &fwq_visitante.Password, &fwq_visitante.Posicionx,
            &fwq_visitante.Posiciony, &fwq_visitante.Destinox, &fwq_visitante.Destinoy,
            &fwq_visitante.DentroParque, &fwq_visitante.IdParque, &fwq_visitante.Parque); err != nil {
            return visitantes, err
        }

        //Vamos añadiendo los visitantes al array
        visitantes = append(visitantes, fwq_visitante)
    }

    if err = results.Err(); err != nil {
        return visitantes, err
    }

    return visitantes, nil
}
```

Ahora procesamos el movimiento del visitante para poder actualizar el mapa con la función `mueveVisitante`:

```
/* Función que modifica las posiciones de los visitantes en el parque en base a sus movimientos */
func mueveVisitante(db *sql.DB, id, movimiento string, visitantes []visitante) {

    var nuevaPosicionX int
    var nuevaPosicionY int

    for i := 0; i < len(visitantes); i++ {

        if id == visitantes[i].ID { // Modificamos la posición del visitante recibido por kafka

            switch movimiento {
            case "N":
                visitantes[i].Posicionx--
            case "S":
                visitantes[i].Posicionx++
            case "W":
                visitantes[i].Posiciony--
            case "E":
                visitantes[i].Posiciony++
            case "NW":
                visitantes[i].Posicionx--
                visitantes[i].Posiciony--
            case "NE":
                visitantes[i].Posicionx--
                visitantes[i].Posiciony++
            case "SW":
                visitantes[i].Posicionx++
                visitantes[i].Posiciony--
            case "SE":
                visitantes[i].Posicionx++
                visitantes[i].Posiciony++
            }

            if visitantes[i].Posicionx == -1 {
                visitantes[i].Posicionx = 19
            } else if visitantes[i].Posicionx == 20 {
                visitantes[i].Posicionx = 0
            }

            if visitantes[i].Posiciony == -1 {
                visitantes[i].Posiciony = 19
            } else if visitantes[i].Posiciony == 20 {
                visitantes[i].Posiciony = 0
            }

            nuevaPosicionX = visitantes[i].Posicionx
            nuevaPosicionY = visitantes[i].Posiciony
        }
    }

    // MODIFICAMOS la posición de dicho visitante en la BD
    // Preparamos para prevenir inyecciones SQL
    sentenciaPreparada, err := db.Prepare("UPDATE visitante SET posicionx = ?, posiciony = ? WHERE id = ?")
    if err != nil {
        panic("Error al preparar la sentencia de modificación: " + err.Error())
    }

    defer sentenciaPreparada.Close()

    // Ejecutar sentencia, un valor por cada '?'
    _, err = sentenciaPreparada.Exec(nuevaPosicionX, nuevaPosicionY, id)
    if err != nil {
        panic("Error al modificar la posición del visitante en la BD: " + err.Error())
    }
}
```

En esta función dependiendo del movimiento recibido actualizaremos las coordenadas del visitante en el mapa en consecuencia, teniendo en cuenta que si se alcanza un valor límite y al ser un mapa circular tenemos que actualizar la coordenada por la del extremo contrario.

Tras esto y volviendo a la segunda casuística, volvemos a obtener los visitantes del parque actualizados y ahora generemos/actualizamos el mapa con la función asignacionPosiciones:

```
/*
 * Función que forma el mapa del parque conteniendo a los visitantes y las atracciones
 * @return [20][20]string : Matriz bidimensional representando el mapa
 */
func [asignacionPosiciones](visitantes []visitante, atracciones []atraccion, mapa [20][20]string) [20][20]string {

    //Asignamos los id de los visitantes
    for i := 0; i < len(mapa); i++ {
        for j := 0; j < len(mapa[i]); j++ {
            for k := 0; k < len(visitantes); k++ {
                if i == visitantes[k].Posicionx && j == visitantes[k].Posiciony && visitantes[k].DentroParque == 1 {
                    mapa[i][j] = visitantes[k].IdParque + "|"
                }
            }
        }
    }

    //Asignamos los valores de tiempo de espera de las atracciones
    for i := 0; i < len(mapa); i++ {
        for j := 0; j < len(mapa[i]); j++ {
            for k := 0; k < len(atracciones); k++ {
                if i == atracciones[k].Posicionx && j == atracciones[k].Posiciony {
                    mapa[i][j] = strconv.Itoa(atracciones[k].TiempoEspera) + "|"
                }
            }
        }
    }

    // Las casillas del mapa que no tengan ni visitantes ni atracciones las representamos con una guion
    for i := 0; i < len(mapa); i++ {
        for j := 0; j < len(mapa[i]); j++ {
            if len(mapa[i][j]) == 0 {
                mapa[i][j] = "-" + "|"
            }
        }
    }
    return mapa
}
```

Tras obtener el mapa devuelto por esta función lo procesamos con un par de bucles for para generar los espacios correspondientes a las filas y después lo convertimos a formato json para enviarlo al visitante utilizando para ello la función productorMapa:

```

/* Función que envia el mapa a los visitantes */
func productorMapa(IpBroker, PuertoBroker string, ctx context.Context, mapa []byte) {
    var brokerAddress string = IpBroker + ":" + PuertoBroker
    var topic string = "movimiento-mapa"

    w := kafka.NewWriter(kafka.WriterConfig{
        Brokers:          []string{brokerAddress},
        Topic:            topic,
        CompressionCodec: kafka.Snappy.Codec(),
    })

    err := w.WriteMessages(ctx, kafka.Message{
        Key:   []byte("Key-Mapa"), //[]byte(strconv.Itoa(i)),
        Value: []byte(mapa),
    })
    if err != nil {
        panic("No se puede mandar el mapa: " + err.Error())
    }
}

```

Por el contrario, si el alias indicado en la petición de movimiento no es válido entonces:

```

} else { // Si el alias no pertenece a un visitante del parque
    respuesta += alias + ":" + "Parque cerrado"
    productorLogin(IpKafka, PuertoKafka, ctx, respuesta)
    results.Close()
}

```

Notificamos al engine utilizando la función de productorLogin vista anteriormente, indicando que el parque está cerrado.

```

else if peticion == "OUT" { // Si se nos ha solicitado una salida del parque

    // Sacamos del parque al visitante y reiniciamos tanto su posición actual como su destino
    sentenciaPreparada, err := db.Prepare("UPDATE visitante SET dentroParque = 0, posicionx = 0, posiciony = 0, destinox = -1, destinoy = -1 WHERE id = ?")
    if err != nil {
        panic("Error al preparar la sentencia de modificación: " + err.Error())
    }

    // Ejecutar sentencia, un valor por cada '?'
    _, err = sentenciaPreparada.Exec(v.ID)
    if err != nil {
        panic("Error al actualizar el estado del visitante respecto al parque: " + err.Error())
    }

    /*encontrado := false

    for i := 0; i < len(visitantesDelEngine) && !encontrado; i++ {
        if visitantesDelEngine[i] == v.ID {
            visitantesDelEngine = remove(visitantesDelEngine, i)
            encontrado = true
        }
    }*/

    sentenciaPreparada.Close()
}

```

La tercera casuística de la función consumidorEngine es aquella en la que se recibe una petición de salida del parque por parte de un visitante, para lo cual reiniciamos la información de dicho visitante en la BD sin más.

```

    } else { // si las credenciales enviadas para iniciar sesión no son válidas

        if parqueLleno(db, maxVisitantes) {
            respuesta += alias + ":" + "Aforo al completo"
            productorLogin(IpKafka, PuertoKafka, ctx, respuesta)
        } else {
            respuesta += alias + ":" + "Parque cerrado"
            productorLogin(IpKafka, PuertoKafka, ctx, respuesta)
        }
    }

    results.Close()
}

}

```

Y la cuarta y última casuística de la función consumidorEngine es aquella en la que las credenciales de login no son válidas o bien el aforo está completo, enviando al visitante un mensaje informando del error/situación con la función productorLogin ya vista.

Continuando con el código del main, tenemos declarada una función anónima y asíncrona que nos va a permitir notificar a los visitantes conectados la caída del engine:

```

c := make(chan os.Signal, 1)
signal.Notify(c, os.Interrupt)
go func() {
    for sig := range c {
        log.Printf("captured %v, stopping profiler and exiting..", sig)
        mensaje := "Engine no disponible"
        mensajeJson, err := json.Marshal(mensaje)
        if err != nil {
            fmt.Println("Error a la hora de codificar el mensaje: %v", err)
        }
        productorMapa(IpKafka, PuertoKafka, ctx, mensajeJson)

        fmt.Println()
        fmt.Println("Engine apagado manualmente")
        pprof.StopCPUProfile()
        os.Exit(1)
    }
}()

```

Para gestionar esta situación hemos hecho uso de los channels de golang y simplemente comentar que la notificación la enviamos por medio de la función ya vista, productorMapa.

```

for {
    visitantesRegistrados, _ = obtenerVisitantesBD(conn) // Obtenemos los visitantes registrados actualmente
    fmt.Println("***** FUN WITH QUEUES RESORT ACTIVITY MAP *****")
    fmt.Println("ID      " + " Nombre      " + " Pos.      " + " Destino      " + " DentroParque")
    //Hay que usar la función TrimSpace porque al parecer tras la obtención de valores de BD se agrega un retorno de carro a cada variable
    //Mostramos los visitantes registrados en la aplicación actualmente
    for i := 0; i < len(visitantesRegistrados); i++ {
        fmt.Println(strings.TrimSpace(visitantesRegistrados[i].ID) + "      " + strings.TrimSpace(visitantesRegistrados[i].Nombre) +
                    "      " + "      (" + strings.TrimSpace(strconv.Itoa(visitantesRegistrados[i].Posicionx)) + "," + strings.TrimSpace(strconv.Itoa(visitantesRegistrados[i].Posiciony)) +
                    ")" + "      " + "      (" + strings.TrimSpace(strconv.Itoa(visitantesRegistrados[i].Destinox)) + "," + strings.TrimSpace(strconv.Itoa(visitantesRegistrados[i].Destinoy)) +
                    ")" + "      " + strings.TrimSpace(strconv.Itoa(visitantesRegistrados[i].DentroParque)))
    }
    fmt.Println() // Para mejorar la visualización

    // Cada X segundos se conectará al servidor de tiempos para actualizar los tiempos de espera de las atracciones
    time.Sleep(time.Duration(5 * time.Second))
    conexionTiempoEspera(conn, IpFWQWaiting, PuertoWaiting)

    fmt.Println() // Para mejorar la visualización
}

```

Para terminar con el main tenemos un bucle for en el que iremos mostrando de forma actualizada el estado de los visitantes que se encuentran registrados en la aplicación/el parque y tras esto hacemos una espera de 5 segundos para después realizar una petición al servidor de tiempos en la que solicitamos que se nos envíen los tiempos de espera de las atracciones del parque actualizados. Esto lo hacemos con la función `conexionTiempoEspera`:

```

/*
* Función que se conecta al servidor de tiempos para obtener los tiempos de espera actualizados
*/
func conexionTiempoEspera(db *sql.DB, IpFWQWaiting, PuertoWaiting string) {

    fmt.Println() // Por limpieza
    fmt.Println("****Conexión con el servidor de tiempo de espera****")
    //fmt.Println("Arrancando el engine para atender los tiempos en el puerto: " + IpFWQWaiting + ":" + PuertoWaiting)
    var connType string = "tcp"
    conn, err := net.Dial(connType, IpFWQWaiting+":"+PuertoWaiting)

    if err != nil {
        fmt.Println("ERROR: El servidor de tiempos de espera no está disponible", err.Error())
    } else {
        fmt.Println("****Actualizando los tiempos de espera****")
        fmt.Println() // Por limpieza

        conn.Write([]byte("Mándame los tiempos de espera actualizados" + "\n")) // Mandamos la petición
        tiemposEspera, _ := bufio.NewReader(conn).ReadString('\n') // Obtenemos los tiempos de espera actualizados

        arrayTiemposEspera := strings.Split(tiemposEspera[:len(tiemposEspera)-1], "|")

        // Actualizamos los tiempos de espera de las atracciones en la BD
        actualizaTiemposEsperaBD(db, arrayTiemposEspera)

        if tiemposEspera != "" {
            log.Println("Tiempos de espera actualizados: " + tiemposEspera)
        } else {
            log.Println("Servidor de tiempos no disponible.")
        }
    }
}

```

En esta función nos conectamos vía socket con el servidor de tiempos de espera y dependiendo de si este está disponible o no actuaremos en consecuencia.

Si no está disponible entonces simplemente mostramos un error por pantalla indicando dicha situación sin hacer nada más. Por el contrario, si se encuentra disponible, entonces mandamos la petición vía socket y leemos la respuesta del servidor, con la cual,

actualizamos los tiempos de espera de la atracciones en la base de datos, utilizando para ello la función actualizaTiemposEsperaBD:

```
/* Función que actualiza los tiempos de espera de las atracciones en la BD */
func actualizaTiemposEsperaBD(db *sql.DB, tiemposEspera []string) {
    results, err := db.Query("SELECT * FROM atraccion")

    // Comprobamos que no se produzcan errores al hacer la consulta
    if err != nil {
        panic("Error al hacer la consulta a la BD: " + err.Error())
    }

    defer results.Close() // Nos aseguramos de cerrar/

    i := 0

    // Recorremos todas las filas de la consulta
    for results.Next() {

        // Preparamos para prevenir inyecciones SQL
        sentenciaPreparada, err := db.Prepare("UPDATE atraccion SET tiempoEspera = ? WHERE id = ?")
        if err != nil {
            panic("Error al preparar la sentencia de modificación: " + err.Error())
        }

        defer sentenciaPreparada.Close()

        infoAtraccion := strings.Split(tiemposEspera[i], ":") // Extraemos el id y el tiempo de espera de la atracción
        idAtraccion := infoAtraccion[0]

        nuevoTiempo, err := strconv.Atoi(infoAtraccion[1])

        if err != nil {
            panic("Error al convertir la cadena con el nuevo tiempo de la atracción")
        }

        // Ejecutar sentencia, un valor por cada '?'
        _, err = sentenciaPreparada.Exec(nuevoTiempo, idAtraccion)
        if err != nil {
            panic("Error al modificar el tiempo de espera de la atracción: " + err.Error())
        }

        i++
    }
}
```

Entonces si todo ha ido bien, se mostrará un mensaje por pantalla informando al respecto y en caso contrario lo mismo.

Guía de despliegue de la aplicación

Lo primero que haremos será conectarnos a una red fiable.

Después comprobaremos las ips de cada una de las máquinas que van a ser utilizadas para el experimento.

Los puertos utilizados son los siguientes

- 9092 kafka
- 9093 registry
- 9094 servidor de tiempos de espera

Finalmente se levantara primero el engine y el registry con la ip de la máquina y los puertos nombrados anteriormente. Después levantamos el gestor de colas y el servidor de tiempos también con la ip y el puerto de espera y finalmente arrancaremos los visitantes y las atracciones indicando la ip y el puerto asociado tal y como se especifica en el enunciado de la práctica.

En golang para poner en marcha alguna aplicación tenemos que ir a la carpeta de cada uno de los programas.

Para compilar ejecutamos el siguiente comando:

```
go build fwq_engine.go
```

Finalmente para ejecutar el programa escribimos el siguiente comando.

```
./fwq_engine 127.0.0.1 9092 30 127.0.0.1 9094
```

Capturas de pantalla que muestran el funcionamiento de las distintas aplicaciones conectadas

Sensor en funcionamiento conectado a la atracción2:

```
jose@asus-jose:~/go/src/github.com/wilmer799/SDPracticas/fwq_Sensor$ ./fwq_sensor localhost 9092 atraccion2
Sensor creado para la atracción (atraccion2) en la que inicialmente hay 58 personas en cola
En la atracción [atraccion2] hay 2 personas en cola
En la atracción [atraccion2] hay 45 personas en cola
En la atracción [atraccion2] hay 60 personas en cola
En la atracción [atraccion2] hay 45 personas en cola
En la atracción [atraccion2] hay 5 personas en cola
En la atracción [atraccion2] hay 9 personas en cola
En la atracción [atraccion2] hay 20 personas en cola
En la atracción [atraccion2] hay 27 personas en cola
En la atracción [atraccion2] hay 60 personas en cola
En la atracción [atraccion2] hay 50 personas en cola
En la atracción [atraccion2] hay 3 personas en cola
En la atracción [atraccion2] hay 56 personas en cola
En la atracción [atraccion2] hay 1 personas en cola
En la atracción [atraccion2] hay 22 personas en cola
En la atracción [atraccion2] hay 44 personas en cola
En la atracción [atraccion2] hay 43 personas en cola
En la atracción [atraccion2] hay 16 personas en cola
[]
```

Servidor de tiempos de espera sin recibir peticiones del engine:

```
jose@jose-B450-AORUS-PRO:~/go/src/github.com/wilmer799/SDPracticas/fwq_WaitingTimeServer$ ./fwq_waitingTimeServer localhost 9094 localhost 9092
Servidor de tiempos atendiendo en localhost:9094
```

Servidor de tiempos de espera recibiendo peticiones del engine y la información proporcionada por el sensor asociado a la atracción 2:

```
jose@asus-jose:~/go/src/github.com/wilmer799/SDPracticas/fwq_WaitingTimeServer$ ./fwq_waitingTimeServer localhost 9094 localhost 9092
Servidor de tiempos atendiendo en localhost:9094
2021/12/21 18:19:28 Cliente engine 127.0.0.1:53148 conectado.
Petición del Engine: Mándame los tiempos de espera actualizados

[ atraccion1:52 personas en cola ]
[ atraccion1:5 personas en cola ]
[ atraccion1:42 personas en cola ]
[ atraccion1:17 personas en cola ]
[ atraccion1:41 personas en cola ]
[ atraccion1:11 personas en cola ]
[ atraccion1:54 personas en cola ]
[ atraccion1:41 personas en cola ]
[ atraccion1:28 personas en cola ]
[ atraccion1:23 personas en cola ]
[ atraccion1:43 personas en cola ]
[ atraccion1:56 personas en cola ]
[ atraccion1:9 personas en cola ]
```

Si cae un sensor el servidor de tiempos de espera recibe una notificación al respecto, por ejemplo si arrancamos un sensor asociado a la atraccion1 y lo tumbamos se recibe el siguiente mensaje en el servidor de tiempos de espera:

```
[ atraccion1:30 personas en cola ]
[ atraccion1:29 personas en cola ]
[ atraccion1:58 personas en cola ]

Sensor de la atracción (atraccion1) desconectado

[ atraccion1:37 personas en cola ]
[ atraccion1:7 personas en cola ]
[ atraccion1:28 personas en cola ]
[ atraccion1:18 personas en cola ]
[ atraccion1:28 personas en cola ]
[ atraccion1:60 personas en cola ]
```

Ahora vamos con el engine, si consultamos el valor de tiempo de espera recibido para la atracción 2 veremos como va actualizándose:

```
***** FUN WITH QUEUES RESORT ACTIVITY MAP ***
ID           Nombre          Pos.

***Conexión con el servidor de tiempo de espera***
***Actualizando los tiempos de espera***

2021/12/21 18:21:19 Tiempos de espera actualizados:
4:40|atraccion15:74|atraccion16:23|atraccion2:36|atraccion9:20|
```

```
***** FUN WITH QUEUES RESORT ACTIVITY MAP ***
ID           Nombre          Pos.

***Conexión con el servidor de tiempo de espera***
***Actualizando los tiempos de espera***

2021/12/21 18:21:24 Tiempos de espera actualizados:
4:40|atraccion15:74|atraccion16:23|atraccion2:9|atraccion9:20|
```

```
***** FUN WITH QUEUES RESORT ACTIVITY MAP ***
ID           Nombre          Pos.

***Conexión con el servidor de tiempo de espera***
***Actualizando los tiempos de espera***

2021/12/21 18:21:29 Tiempos de espera actualizados:
4:40|atraccion15:74|atraccion16:23|atraccion2:108|atraccion9:20|
```

Ahora vamos a arrancar el Registry y vamos a ver lo que se muestra antes de recibir alguna petición por parte de un visitante:

```
jose@asus-jose:~/go/src/github.com/wilmer799/SDPracticas/FWQ_Registry$ ./fwq_registry localhost 9093
Arrancando el Registry, atendiendo en localhost:9093
```

Entonces, vamos a arrancar un visitante y a solicitar un registro para ver cómo se comportan tanto el registry como el engine.

```
jose@asus-jose:~/go/src/github.com/wilmer799/SDPracticas/FWQ_Registry$ ./fwq
 _registry localhost 9093
Arrancando el Registry, atendiendo en localhost:9093
2021/12/21 18:27:49 Visitante 127.0.0.1:39812 conectado.
Visitante a registrar -> ID: carlos123 | Nombre: carlos | Password: 1234
Visitante 127.0.0.1:39812 desconectado.
■

jose@asus-jose:~/go/src/github.com/wilmer799/SDPracticas/FWQ_Visitor$ ./fwq_
visitor localhost 9093 localhost 9092
**Bienvenido al parque de atracciones**

***Menu parque de atracciones***
1.Crear perfil
2.Editar perfil
3.Moverse por el parque
Elige la acción a realizar:
*****Creación de perfil*****
Introduce tu ID:carlos123
Introduce tu nombre:carlos
Introduce tu contraseña:1234
2021/12/21 18:27:55 Respuesta del Registry: Visitante registrado en el parqu
e. Actualmente hay 1 visitantes registrados.
***Menu parque de atracciones***
1.Crear perfil
2.Editar perfil
3.Moverse por el parque
Elige la acción a realizar:■
```

Y si consultamos el engine:

Vemos como el visitante al no haber iniciado sesión en el parque se muestra con la información por defecto inicial.

Si un visitante cae en medio de un proceso de registro en el registry únicamente se mostrará lo siguiente:

2021/12/21 18:31:26 Visitante 127.0.0.1:42098 conectado.
Visitante 127.0.0.1:42098 desconectado.

Ahora vamos a probar a realizar un update desde el visitante sobre el registry:

```
jose@asus-jose:~/go/src/github.com/wilmer799/SDPracticas/fwq_Registry$ ./fwq_
_registry localhost 9093
Arrancando el Registry, atendiendo en localhost:9093
2021/12/21 18:36:54 Visitante 127.0.0.1:46828 conectado.
Visitante a editar -> ID: carlos123 | Nombre: charlie | Password: 12345
Visitante 127.0.0.1:46828 desconectado.
[]

jose@asus-jose:~/go/src/github.com/wilmer799/SDPracticas/fwq_Visitor$ ./fwq_
visitor localhost 9093 localhost 9092
**Bienvenido al parque de atracciones**
***Menu parque de atracciones***
1.Crear perfil
2.Editar perfil
3.Moverse por el parque
Elige la acción a realizar:2
Has entrado a editar perfil
Información del visitante que se quiere modificar:
Introduce el ID:carlos123
Introduce el nombre:charlie
Introduce la contraseña:12345
2021/12/21 18:37:26 Respuesta del Registry: Visitante actualizado correctamente
***Menu parque de atracciones***
1.Crear perfil
2.Editar perfil
3.Moverse por el parque
Elige la acción a realizar:[]
```

Por lo tanto, ahora al consultar la información mostrada en el engine veremos al visitante con la información actualizada:

```
***Conexión con el servidor de tiempo de espera***
***Actualizando los tiempos de espera***

2021/12/21 18:39:03 Tiempos de espera actualizados: atraccion1:85|atraccion10:13|atra
atraccion16:23|atraccion2:63|atraccion3:15|atraccion4:65|atraccion5:10|atraccion6:40

***** FUN WITH QUEUES RESORT ACTIVITY MAP *****
ID           Nombre      Pos.        Destino      DentroParque
carlos123    charlie    (0,0)      (-1,-1)     0
```

Una posibilidad de puede ocurrir es que el visitante trate de realizar un registro o un update y el registry no esté disponible, en tal caso se mostrará el siguiente mensaje en el visitante:

```
jose@asus-jose:~/go/src/github.com/wilmer799/SDPracticas/fwq_Registry$ []
jose@asus-jose:~/go/src/github.com/wilmer799/SDPracticas/fwq_Visitor$ ./fwq_
visitor localhost 9093 localhost 9092
**Bienvenido al parque de atracciones**

***Menu parque de atracciones***
1.Crear perfil
2.Editar perfil
3.Moverse por el parque
Elige la acción a realizar:1
*****Creación de perfil*****
Error al conectarse al Registry: dial tcp 127.0.0.1:9093: connect: connection refused
***Menu parque de atracciones***
1.Crear perfil
2.Editar perfil
3.Moverse por el parque
Elige la acción a realizar:[]
```

Otro caso que puede ocurrir es que el engine trate de conectarse a un servidor de tiempos de espera que no está disponible, en cuyo caso se mostrará esta situación adecuadamente por pantalla durante la ejecución del engine:

```
***Conexión con el servidor de tiempo de espera***
ERROR: El servidor de tiempos de espera no está disponible dial tcp 127.0.0.1:9094: connect: connection refused

***** FUN WITH QUEUES RESORT ACTIVITY MAP *****
ID           Nombre      Pos.        Destino      DentroParque
carlos123    charlie    (0,0)      (-1,-1)     0
```

Ahora vamos a probar desde el visitante la última de las opciones, que es la entrada al parque:

```
***Menu parque de atracciones***
1.Crear perfil
2.Editar perfil
3.Moverse por el parque
Elige la acción a realizar:3
*Bienvenido al parque de atracciones*
Por favor introduce tu alias:carlos123
y tu password:12345
Enviando credenciales -> carlos123:12345:-1,-1
Respuesta del engine: carlos123:Acceso concedido
El visitante está dentro del parque
Enviando movimiento: carlos123:IN:-1,-1
c| - | - | - | - | - | - | - | - | - | - | - | - | - | - |
- | - | - | 99 | - | - | - | - | - | - | - | - | - | - | - |
- | - | - | 90 | - | - | - | - | - | - | - | - | - | - | - |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 40 |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 74 |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 17 |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 15 |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 20 |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 80 |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 10 |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 85 |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 65 |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 23 |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 77 |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 7 |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 13 |
- | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 40 |
```

Me dirijo a la atracción con tiempo de espera igual a 23

Enviando movimiento: carlos123:NW:15,15

```
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-| 99 | - | - | - | - | - | - | - | - | - | - | - |
-|-|-|-| 90 | - | - | - | - | - | - | - | - | - | - | - | - |
```

Si consultamos el engine veremos el estado del visitante cambiando en cada iteración:

```
***** FUN WITH QUEUES RESORT ACTIVITY MAP *****
ID           Nombre      Pos.        Destino      DentroParque
carlos123    charlie    (5,16)     (6,5)       1

***Conexión con el servidor de tiempo de espera***
***Actualizando los tiempos de espera***

2021/12/21 18:41:49 Tiempos de espera actualizados: atraccion1:85|atraccion10:13|atra
atraccion16:23|atraccion2:72|atraccion3:15|atraccion4:65|atraccion5:10|atraccion6:40

***** FUN WITH QUEUES RESORT ACTIVITY MAP *****
ID           Nombre      Pos.        Destino      DentroParque
carlos123    charlie    (6,14)     (6,5)       1

***Conexión con el servidor de tiempo de espera***
***Actualizando los tiempos de espera***

2021/12/21 18:41:54 Tiempos de espera actualizados: atraccion1:85|atraccion10:13|atra
atraccion16:23|atraccion2:27|atraccion3:15|atraccion4:65|atraccion5:10|atraccion6:40

***** FUN WITH QUEUES RESORT ACTIVITY MAP *****
ID           Nombre      Pos.        Destino      DentroParque
carlos123    charlie    (6,12)     (6,5)       1

***Conexión con el servidor de tiempo de espera***
***Actualizando los tiempos de espera***

2021/12/21 18:41:59 Tiempos de espera actualizados: atraccion1:85|atraccion10:13|atra
atraccion16:23|atraccion2:117|atraccion3:15|atraccion4:65|atraccion5:10|atraccion6:40

***** FUN WITH QUEUES RESORT ACTIVITY MAP *****
ID           Nombre      Pos.        Destino      DentroParque
carlos123    charlie    (6,11)     (6,5)       1
```

Démonos cuenta también, que si nos fijamos en los mapas recibidos en el visitante podemos ver cómo se va actualizando la posición del visitante (representado con una c) y el tiempo de espera de la atracción 2 que en esta captura pasa de tener un tiempo de espera de 99 a un tiempo de espera de 108:

```
Enviando movimiento: carlos123:S:3,18
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|c|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|40|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|74|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|17|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|15|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|20|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|80|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|10|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|85|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|65|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|23|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|77|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|7|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|13|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|40|-|
Enviando movimiento: carlos123:S:3,18
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|c|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|108|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|90|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|40|-|
```

Otra situación que se puede dar en el visitante es el caso en que mandamos credenciales inválidas:

```
**Bienvenido al parque de atracciones**

***Menu parque de atracciones***
1.Crear perfil
2.Editar perfil
3.Moverse por el parque
Elige la acción a realizar:3
*Bienvenido al parque de atracciones*
Por favor introduce tu alias:jose123
y tu password:1234
Enviando credenciales -> jose123:1234:-1,-1
Respuesta del engine: jose123:Parque cerrado
Parque cerrado
***Menu parque de atracciones***
1.Crear perfil
2.Editar perfil
3.Moverse por el parque
Elige la acción a realizar:■
```

En este caso, el visitante recibe un mensaje por parte del servidor informando de que el parque está cerrado para él. Algo parecido ocurre cuando el aforo del parque está completo:

```
***Menu parque de atracciones***
1.Crear perfil
2.Editar perfil
3.Moverse por el parque
Elige la acción a realizar:3
*Bienvenido al parque de atracciones*
Por favor introduce tu alias:marco123
y tu password:1234
Enviando credenciales -> marco123:1234:-1,-1
Respuesta del engine: marco123:Aforo al completo
Aforo al completo
***Menu parque de atracciones***
1.Crear perfil
2.Editar perfil
3.Moverse por el parque
Elige la acción a realizar:■
```

Y vamos con las últimas situaciones destacables que se pueden dar durante la ejecución de la aplicación, la primera de ellas es en la que el engine se cae, en cuyo caso los visitantes que están trabajando con dicho engine serán correspondientemente notificados, lo que provoca que estos vuelvan al menú de su aplicación:

```
***** FUN WITH QUEUES RESORT ACTIVITY MAP *****
ID           Nombre      Pos.       Destino      DentroParque
carlos123    charlie    (9,15)    (9,17)      1
hugo123     hugo       (14,15)   (9,17)      1
jose123      jose       (1,16)    (7,8)       1
marco123    marco      (0,0)     (-1,-1)    0

^C2021/12/21 21:52:10 captured interrupt, stopping profiler and exiting..

Engine apagado manualmente
jose@asus-jose:~/go/src/github.com/wilmer799/SDPPracticas/FWQ_Engine$ █
```

Y ahora si nos vamos uno de los visitantes que se encontraban dentro del parque:

Y vemos como se nos ha devuelto al menú de la aplicación visitante y se nos ha informado que el engine ha dejado de estar disponible.

Para terminar con las capturas de funcionamiento, vamos a mostrar lo que ocurre cuando un visitante decide salir del parque o la aplicación de este se cae:

Y si miramos el engine:

```
***Conexión con el servidor de tiempo de espera***
ERROR: El servidor de tiempos de espera no está disponible dial tcp 127.0.0.1:9094: c

***** FUN WITH QUEUES RESORT ACTIVITY MAP *****
ID           Nombre      Pos.        Destino      DentroParque
jose123     jose        (5,5)       (6,5)        1

***Conexión con el servidor de tiempo de espera***
ERROR: El servidor de tiempos de espera no está disponible dial tcp 127.0.0.1:9094: c

***** FUN WITH QUEUES RESORT ACTIVITY MAP *****
ID           Nombre      Pos.        Destino      DentroParque
jose123     jose        (0,0)       (-1,-1)     0
```

Es fácil notar que los datos del visitante que ha salido del parque se han reiniciado adecuadamente.