Fun With Queues
Memoria

Sergio Baeza Carrasco,53979913V
Oliver Vincent Rice, Y1421890K

Universitat d'Alacant
Universidad de Alicante

Sergio Baeza Carrasco, 53979913V
Oliver Vincent Rice, Y1421890K

ÍNDICE

1. Informe detallado del Software

a. FWQ_Engine

Para la ejecución del **FWQ_Engine.py** vamos a recibir por línea de parámetros 3 parámetros donde tendremos la *IP y Puerto* del gestor de colas, **el número máximo de visitantes** permitidos en el parque y por último la *IP y Puerto* del FWQ_WaitingTimeServer, el gestor de tiempos de espera.

Cómo recibiremos las **IP y Puertos** con el formato de **IP:Puerto** vamos a tener que trocear esta cadena para leer sus valores. Una vez verificamos que tenemos los parámetros requeridos, vamos a conectarnos con la base de datos, luego pedimos la información de las atracciones, su posición y demás.
También vamos a pedir de la base de datos, los Tokens de usuarios que ya estén logeados en el parque. De esta forma podremos hacer que si se apaga el **Engine** y se restaura, los usuarios sigan con su **sesión iniciada**.

Funciones **incorporadas**:

- **checkIfWaitingServerIsOnline():** Esta función verifica si hay una conexión con el servidor y si no, se queda en espera hasta que lo haya.

```
# Funcion que revisa si hay conexion
def checkIfWaitingServerIsOnline():
    while(1):
        if(WTS_isAlive != 1):
            print("[WAITING TIME SERVER] Starting task")
            connectWTS()
        time.sleep(20)
```

- **handleLoginRequest():** Esta función usaremos un topic en kafka llamada “logindetails” y éste se queda como consumidor, a la espera de la información de **Login del usuario/visitante del parque**, una vez nos llega la información, decodificamos el mensaje y comprobamos si hay suficiente espacio en el parque para un visitante nuevo, en caso de que si, pasamos a la función “**generateLoginResponse()**”, si el parque está lleno devolvemos un mensaje de respuesta.

```python
# Funcion que permite el login
def handleLoginRequest():
    #Consumidor de login
    global KAFKA_SERVER
    global MAX_VISITORS
    topic = "logindetails"
    consumer = KafkaConsumer(topic, bootstrap_servers = KAFKA_SERVER)

    print("[LOGIN] Awaiting for info on Kafka Server topic = " + topic)
    for message in consumer:
        valor = message.value.decode(FORMAT)
        valor = valor.split(":")
        time.sleep(1)
        if VISITORS < MAX_VISITORS:
            generateLoginResponse(valor[0], valor[1])
        else:
            print("PARQUE FULL")
            topic = "logintoken"
            producer = KafkaProducer(bootstrap_servers = KAFKA_SERVER)
            send = str(valor[0]) + ":0:" + "MAX_VISITORS_REACHED"
            producer.send(topic, send.encode(FORMAT))
            producer.close()
```

- **generateLoginResponse():** En esta función usaremos el topic de "*logintoken*", que se encarga de **recibir y enviar información** de los Tokens de sesión de cada usuario. Aquí lo que hacemos es realizar una conexión a la base de datos donde realizamos una sentencia SELECT con la condición de que **coincidan los datos del usuario**. Si se verifican dichos datos, incrementamos la cantidad de usuarios en 1, y le asignamos un TOKEN a este usuario en otra tabla de la base de datos. Al Token se le asigna un tiempo de sesión máximo que una vez se agote, se expira dicho Token (esto aún no está implementado). Si todo este proceso falla, devolvemos un mensaje de error que indica que no se ha hecho el Login correspondiente del usuario.

```python
# Update information
def userinfo():
    global users
    consumer = KafkaConsumer('userinfo',bootstrap_servers=KAFKA_SERVER)
    for message in consumer:
        # usuario:token:3,1:1
        valor = message.value.decode(FORMAT)
        # print("He recibido: " + valor)
        valor = valor.split(":")
        user = valor[0]
        token = valor[1]
        coordenada = [valor[2].split(",")[0], valor[2].split(",")[1]]
        atDirige = valor[3]
        conDb = sqlite3.connect(RUTE_DB)
        print("[USER-INFO] Got a request... Updating user info")
        cur = conDb.cursor()
        cur.execute('SELECT * FROM tokens WHERE user = "' + user + '" AND token = "' + token + '"')
        row = cur.fetchone()
        if row is not None:
            if int(row[2]) > int(time.time()):
                # Se encuentra el usuario con la sesion iniciada
                find = False
                i = 0
                for visitor in users:
                    if(visitor[0] == user):
                        find = True
                    if find == False:
                        i = i + 1
                if find == True:
                    users[i][3][0] = int(coordenada[0])
                    users[i][3][1] = int(coordenada[1])
                    users[i][3][2] = int(atDirige)
                    users[i][4] = int(time.time())
                if find == False:
                    users.append([row[0], row[1], row[2], [int(coordenada[0]), int(coordenada[1]), int(atDirige)], int(time.time())])
```

- **userInfo():** Esta función recibe como consumidor, una cadena de tipo **"usuario:token:3,1:1"** donde vamos a verificar que el usuario tiene un Token no expirado y así sabemos que tiene una sesión abierta en el parque, de ser así, comprobamos que el usuario existe y le asignaremos la posición que tenía anteriormente en el mapa. Esto puede ser útil ya que si el servidor da problemas de conexión, los usuarios pueden seguir dentro del parque.

```python
# Update information
def userinfo():
    global users
    consumer = KafkaConsumer('userinfo',bootstrap_servers=KAFKA_SERVER)
    for message in consumer:
        # usuario:token:3,1:1
        valor = message.value.decode(FORMAT)
        # print("He recibido: " + valor)
        valor = valor.split(":")
        user = valor[0]
        token = valor[1]
        coordenada = [valor[2].split(",")[0], valor[2].split(",")[1]]
        atDirige = valor[3]
        conDb = sqlite3.connect(RUTE_DB)
        print("[USER-INFO] Got a request... Updating user info")
        cur = conDb.cursor()
        cur.execute('SELECT * FROM tokens WHERE user = "' + user + '" AND token = "' + token + '"')
        row = cur.fetchone()
        if row is not None:
            if int(row[2]) > int(time.time()):
                # Se encuentra el usuario con la sesion iniciada
                find = False
                i = 0
                for visitor in users:
                    if(visitor[0] == user):
                        find = True
                    if find == False:
                        i = i + 1
                if find == True:
                    users[i][3][0] = int(coordenada[0])
                    users[i][3][1] = int(coordenada[1])
                    users[i][3][2] = int(atDirige)
                    users[i][4] = int(time.time())
                if find == False:
                    users.append([row[0], row[1], row[2], [int(coordenada[0]), int(coordenada[1]), int(atDirige)], int(time.time())])
```

- **sendMap():** Aquí lo que se hace es enviar el mapa con los usuarios ya introducidos dentro de él. Nos conectamos al topic **"mapinfo"**, el mapa se envía en forma de cadena tal que los bucles for que tenemos nos crean el formato a enviar, "Id:tiempo:posX:posY, …. #nombre:atDirige:posX:posY" esta información nos indica la id, tiempo y posición del usuario en el mapa, a su vez, tenemos a **dónde se dirige.**

```python
def sendMap():
    global KAFKA_SERVER
    global attractions
    global users
    producer = KafkaProducer(bootstrap_servers = KAFKA_SERVER)
    topic = "mapinfo"
    while(1):
        # Format at Id:tiempo:posX:posY,...#nombre:atDirige:PosX:posY
        send = ""
        if len(attractions) == 0:
            send = "NONE"
        for i in range(len(attractions)):
            send = send + str(i+1) + ":" + str(attractions[i][2]) + ":" + str(attractions[i][0]) + ":" + str(attractions[i][1])
            if i != len(attractions)-1:
                send = send + ","
        send = send + "#"
        if len(users) == 0:
            send = send + "NONE"
        for i in range(len(users)):
            if(users[i][3] != -1):
                send = send + str(users[i][0]) + ":" + str(users[i][3][2]) + ":" + str(users[i][3][0]) + ":" + str(users[i][3][1])
            else:
                send = send + str(users[i][0]) + ":-1" + ":" + str(users[i][3][0]) + ":" + str(users[i][3][1])
            if i != len(users)-1:
                send = send + ","
        print("[MAP SENDER] Sending map to the topic: " + topic)
        producer.send(topic, send.encode('utf-8'))
        time1 = 5 #random.randint(1, 3)
        time.sleep(time1)
```

- **userKeepAlive():** Esta función determina **si un usuario está en el parque** o si se ha desconectado, en el caso de haberse desconectado (mirando si se ha expirado la sesión) le restamos uno a la cantidad de usuarios del parque.

```python
# Keep Alive
def userKeepAlive():
    global VISITORS
    global users
    while 1:
        el = 0
        for u in users:
            if (int(time.time())-u[4]) > 10:
                print("[KEEP ALIVE] User: " + u[0] + " disconnected")
                el = u
        if el != 0:
            users.remove(el)
            VISITORS = VISITORS - 1
        time.sleep(10)
```

MANEJO API OPENWEATHER

Aquí se realiza la monitorización de los datos recibidos del API de OPENWEATHER dependiendo de las ciudades proporcionadas por el usuario. Éstas se dividen en 4 zonas las cuales repartiremos en el mapa y dependiendo de las temperaturas recibidas por el API serán zonas activas o no.

```
def weatherMonitoring():
    global w_config, cities
    print("[WEATHER] Starting weather monitoring...")
    while(1):
        if w_config != -1:
            api_url = "http://api.openweathermap.org/data/2.5/weather?q=" + w_config[1][0] +"&appid=" + w_config[0]
            response = requests.get(api_url)
            if response.status_code == 200:
                json = response.json()
                temp = float(json["main"]["temp"]) - 273.15
                cities[0] = int(temp)
            else:
                cities[0] = -1

            api_url = "http://api.openweathermap.org/data/2.5/weather?q=" + w_config[1][1] +"&appid=" + w_config[0]
            response = requests.get(api_url)
            if response.status_code == 200:
                json = response.json()
                temp = float(json["main"]["temp"]) - 273.15
                cities[1] = int(temp)
            else:
                cities[1] = -1

            api_url = "http://api.openweathermap.org/data/2.5/weather?q=" + w_config[1][2] +"&appid=" + w_config[0]
            response = requests.get(api_url)
            if response.status_code == 200:
                json = response.json()
                temp = float(json["main"]["temp"]) - 273.15
                cities[2] = int(temp)
            else:
                cities[2] = -1

            api_url = "http://api.openweathermap.org/data/2.5/weather?q=" + w_config[1][3] +"&appid=" + w_config[0]
            response = requests.get(api_url)
            if response.status_code == 200:
                json = response.json()
                temp = float(json["main"]["temp"]) - 273.15
                cities[3] = int(temp)
            else:
                cities[3] = -1

            print("[WEATHER] The weather of the 4 cities is: " + str(cities))
        time.sleep(5)
```

b. FWQ_Registry

Primero vamos a mirar el apartado de **crear un perfil nuevo**, lo primero que haremos será conectarse a la base de datos, luego intentará insertar los datos de usuario que le pasamos desde el módulo de **FWQ_Visitante mediante una conexión cliente servidor de sockets**. Si al realizar el insert en la base de datos el usuario ya existe o hay algún problema, para cada caso lanzaremos una excepción y este código lo enviamos al visitante para que lo interprete.

```
if user_info[0]==0:
    # Crear usuario
    try:
        sqliteConnection = sqlite3.connect('./bd/basededatos.db')
        cursor = sqliteConnection.cursor()
        print("Successfully Connected to SQLite")
        cursor.execute('''insert into users(username,password) values (?,?);''',(user_info[1], user_info[2]))
        sqliteConnection.commit()
        print("USUARIO CREADO")
        conn.send("1".encode(FORMAT))
        cursor.close()
    except sqlite3.Error as error:
        print("USUARIO YA EXISTE")
        conn.send("0".encode(FORMAT))
        print("Error while connecting to sqlite", error)
    finally:
        if sqliteConnection:
            sqliteConnection.close()
            print("The SQLite connection is closed")
```

Para editar el nombre del usuario, primero nos **aseguramos** de que éste tenga una **cuenta existente y lo verificamos**, en caso contrario saltará una excepción.
Si **todo se verifica correctamente**, **actualizamos la fila correspondiente del usuario** con el nombre nuevo, asegurándonos de que no existe ya. En caso de que existiese un **usuario igual**, lanza excepción al visitante donde tendría que volver a empezar con otro nombre diferente.

```
# Editar perfil
if user_info[4]==1:
    # Editar nombre de usuario
    try:
        sqliteConnection = sqlite3.connect('./bd/basededatos.db')
        cursor = sqliteConnection.cursor()
        print("Successfully Connected to SQLite")

        userNew = [user_info[3], user_info[1], user_info[2]]
        cursor.execute("SELECT COUNT(*) FROM users WHERE username = (?) AND password = (?)", (userNew[1],userNew[2]))
        sqliteConnection.commit()
        data = cursor.fetchall()
        print(data)
        if data==[(0,)]:
            raise sqlite3.Error()
        print("USUARIO LOGGEADO")
        cursor.execute("UPDATE users SET username = (?) WHERE username = (?) AND password = (?)", userNew)
        sqliteConnection.commit()
        conn.send("1".encode(FORMAT))
        cursor.close()
    except sqlite3.Error as error:
        print("USUARIO/CONTRASEÑA INCORRECTA")
        conn.send("0".encode(FORMAT))
        print("Error while connecting to sqlite", error)
    finally:
        if sqliteConnection:
            sqliteConnection.close()
            print("The SQLite connection is closed")
```

Por último, **modificar contraseña**, se implementa de una forma parecida al modificar usuario, con una diferencia, de que **chequear que la contraseña** ya existe en la base de datos.

```
# Editar password de usuario
try:
    sqliteConnection = sqlite3.connect('./bd/basededatos.db')
    cursor = sqliteConnection.cursor()
    print("Successfully Connected to SQLite")

    userNew = [user_info[3], user_info[1], user_info[2]]
    cursor.execute("SELECT COUNT(*) FROM users WHERE username = (?) AND password = (?)", (userNew[1],userNew[2]))
    sqliteConnection.commit()
    data = cursor.fetchall()
    print(data)
    if data==[(0,)]:
        raise sqlite3.Error()
    print("USUARIO LOGGEADO")
    cursor.execute("UPDATE users SET password = (?) WHERE username = (?) AND password = (?)", userNew)
    sqliteConnection.commit()
    conn.send("1".encode(FORMAT))
    cursor.close()
except sqlite3.Error as error:
    print("USUARIO/CONTRASEÑA INCORRECTA")
    conn.send("0".encode(FORMAT))
    print("Error while connecting to sqlite", error)
finally:
    if sqliteConnection:
        sqliteConnection.close()
        print("The SQLite connection is closed")
```

c. FWQ_Visitor

Primero que nada, le **mostraremos** a nuestros usuarios **un menú,** donde pueden escoger de las 4 opciones distintas. Si la respuesta es igual a 1, entramos al parque, **pediremos un usuario y contraseña** que luego se enviará al FWQ_Engine para verificar estos datos en la base de datos.
Aquí es donde volvemos a comprobar si el usuario recibe el **token**, **si no hubiera respuesta**, **damos por seguro que el Engine no está disponible**.
Una vez hayamos verificado el usuario nos volveremos a apoyar en el paralelismo de funciones, para tener todo enviando y recibiendo datos al mismo tiempo, pudiendo actualizar el mapa, enviar información del usuario etc.

Funciones **incorporadas**:

- **printMap():** Función que tiene **3 estados**: imprimir **el mapa**, **esperar información** del mapa por parte del servidor y que se **ha perdido la conexión con el servidor**

```
    if TARGET_ATRACCION != -1:
        print("[" + str(ATTRACTIONS[TARGET_ATRACCION][2]+1) + ", " + str(ATTRACTIONS[TARGET_ATRACCION][3]+1) + "]")
    else:
        print("None")
    for u in USERS:
        if u[0] != MY_USERNAME:
            if len(u[0]) > 1:
                print(u[0][0:2] + "\t\t",  end="")
            else:
                print(u[0] + "\t\t", end="")
            print("[" + str(u[2]+1) + ", " + str(u[3]+1) + "]" + "\t\t\t", end="")
            if u[3] != -1:
                print("[" + str(ATTRACTIONS[u[1]][2]+1) + ", " + str(ATTRACTIONS[u[1]][3]+1) + "]")

            else:
                print("None")

elif PARQUE_ONLINE == 0:

    clear = lambda: os.system('cls')
    clear()
    print(Fore.GREEN)
    print("Esperando información del mapa por parte del servidor... " + Fore.RESET)
else:
    clear = lambda: os.system('cls')
    clear()
    print(Fore.RED)
    print("Se ha perdido la conexion con el servidor... Esperando..." + Fore.RESET)

time.sleep(2)
```

- **getTarget():** Como su **nombre indica**, esta función busca el destino de un usuario basándo se en la lista de atracciones que tenemos disponibles. Pondremos como condición elegir entre **los que tienen un tiempo menor** o **igual que 60 minutos**. En caso de que no hayan atracciones disponibles, saltará dicho mensaje por pantalla y **esperaremos.**

```
def getTarget():
    while 1:
        global TARGET_ATRACCION
        if TARGET_ATRACCION == -1:
            # no existe ninguna atraccion
            posibles = list()
            if len(ATTRACTIONS) > 0:
                #hay atracciones
                #print(ATTRACTIONS)
                for a in range(len(ATTRACTIONS)):
                    if ATTRACTIONS[a][1] <= 60 and ATTRACTIONS[a][1] != -1:
                        #print("añado la " + str(a))
                        posibles.append(a)
            if len(posibles) <= 0:
                print("No hay atracciones disponibles")
            else:
                don = random.randint(0, (len(posibles)-1))
                TARGET_ATRACCION = posibles[don]
        else:
            if(ATTRACTIONS[TARGET_ATRACCION-1][1] > 60):
                # ha aumentado el tiempo
                TARGET_ATRACCION = -1
        time.sleep(5)
```

- **move():** Esta función **realiza el movimiento del visitante** por el parque, más concretamente, hacia su próximo destino.
  Primero nos aseguramos de que **el parque sigue en línea** y que tenemos **marcado un destino**.
  Luego vamos viendo las **posiciones adyacentes al visitante** y elegimos el que más cerca del destino esté, tomando éste como posición siguiente. Más adelante esta información **se envía al Engine** para meterlo en el mapa.

```
def move():
    global ATRACTION_ACTUAL
    global COORDENADAS_ACTUALES
    global ATTRACTIONS
    global TARGET_ATRACCION
    while 1:
        if TARGET_ATRACCION != -1 and PARQUE_ONLINE != -1:
            # suponemos que hay una atraccion.
            aX = ATTRACTIONS[TARGET_ATRACCION][2]
            aY = ATTRACTIONS[TARGET_ATRACCION][3]
            cX = COORDENADAS_ACTUALES[0]
            cY = COORDENADAS_ACTUALES[1]
            newX = 0
            newY = 0
            if aY < cY:
                newY = -1
            if aY > cY:
                newY = +1
            if aX < cX:
                newX = -1
            if aX > cX:
                newX = +1
            if(newX == 0) and newY == 0:
                #Estamos en la posicion de la atraccion
                ATRACTION_ACTUAL = TARGET_ATRACCION
                TARGET_ATRACCION = -1
                time.sleep(10)

            COORDENADAS_ACTUALES[0] = cX + newX
            COORDENADAS_ACTUALES[1] = cY + newY

        time.sleep(2)
```

- **sendInfoEngine():** Aquí creamos **un productor** que envía toda la información del usuario a un **topic llamado "userinfo"** que vimos que se consumía en el FWQ_Engine. Enviaremos **el nombre de usuario, su token y las coordenadas actuales y destino.**

```
def sendInfoToEngine():
    producer = KafkaProducer(bootstrap_servers = KAFKA_SERVER)
    while 1:
        # usuario:token:3,1:1
        send = MY_USERNAME + ":" + MY_TOKEN + ":" + str(COORDENADAS_ACTUALES[0]) + "," + str(COORDENADAS_ACTUALES[1]) + ":" + str(TARGET_ATRACCION)
        producer.send("userinfo", send.encode(FORMAT))
        time.sleep(1)
```

- **keepAlivePark():** Aquí determinamos el **estado del parque**, con **timestamps** actuales y anteriores averiguamos si sigue en línea el servidor o no.

```
def keepAlivePark():
    global PARQUE_ONLINE, LAST_TIMESTAMP
    while 1:
        ahora = int(time.time())
        resta = ahora - LAST_TIMESTAMP
        if(LAST_TIMESTAMP == -1):
            resta = 0
        if(resta > 10):
            PARQUE_ONLINE = -1
        time.sleep(3)
```

El visitante también se encarga de decir al registro **si quiere modificar su información o crear un perfil nuevo:**

REGISTRO

```
def registro(host, port):
    try:
        ADDR_REGISTRO = (host, port)
        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client.connect(ADDR_REGISTRO)

        print (f"Establecida conexión en [{ADDR_REGISTRO}]")


        print("Nombre de usuario: ")
        usuario = input()
        contra = "1"
        password = "2"
        while contra != password:
            print("Contraseña: ")
            password = input()
            print("Repetir contraseña: ")
            contra = input()
            if(contra != password):
                print("Las contraseñas no coinciden")

        user_info = [0, usuario, password]
        socketSend(client, str(user_info))

        if(client.recv(2048).decode(FORMAT) == "1"):
            print("USUARIO CREADO")
        elif client.recv(2048).decode(FORMAT) == "0":
            print("USUARIO YA EXISTE")

    except:
        print("Error al conectar con el servidor de Registro")
    client.close()
```

Sergio Baeza Carrasco, 53979913V
Oliver Vincent Rice, Y1421890K

EDITAR NOMBRE DE USUARIO

```python
def editUser(host, port):
    ADDR_REGISTRO = (host, port)

    try:
        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client.connect(ADDR_REGISTRO)
        print (f"Establecida conexión en [{ADDR_REGISTRO}]")


        user = "incorrect"

        while user != "correct" :
            print("")
            print("¿Qué desea modificar?")
            print("1 - Nombre de usuario")
            print("2 - Contraseña")
            print("3 - Salir")

            option = input()


            if option=='1':

                print("Nombre de usuario: ")
                usuario = input()
                print("Contraseña: ")
                password = input()
                print("Nombre de usuario nuevo: ")
                new_username = input()
                user_info = [1, usuario, password, new_username, 1]
                # print("Envio al servidor: ", usuario, password, new_username)

                socketSend(client, str(user_info))

                recibido = client.recv(2048).decode(FORMAT)

                if (recibido == "1"):
                    print("PERFIL ACTUALIZADO")

                elif (recibido == "0"):
                    print("USUARIO/CONTRASEÑA INCORRECTA")
```

MODIFICAR CONTRASEÑA

```python
    elif option=='2':

        contra = ""
        password_new = "/"
        print("Nombre de usuario: ")
        usuario = input()
        print("Contraseña: ")
        password = input()

        while contra != password_new:
            print("Contraseña nueva: ")
            contra = input()
            print("Repetir contraseña:")
            password_new = input()
            if(contra != password_new):
                print("Las contraseñas no coinciden")



        user_info = [1, usuario, password, password_new, 2]
        print("Envio al servidor: ", usuario, password, password_new)
        socketSend(client, str(user_info))
        recibido2 = client.recv(2048).decode(FORMAT)

        if (recibido2 == "1"):
            print("PERFIL ACTUALIZADO")


        elif (recibido2 == "0"):
            print("USUARIO/CONTRASEÑA INCORRECTA")
    elif option == "3":
        user = "correct"

int("Error al conectar con el servidor de Registro")
```

Sergio Baeza Carrasco, 53979913V
Oliver Vincent Rice, Y1421890K

FUNCIONES NUEVOS QUE CONSUMEN EL API_REST

```
    if option=='1':

        print("Nombre de usuario: ")
        usuario = input()
        print("Contraseña: ")
        password = input()
        print("Nombre de usuario nuevo: ")
        new_username = input()

        response = requests.put("https://127.0.0.1:5000/users", json={"option":option,"newname":new_username,"name":usuario,"email":password}, verify='cert.pem')
        print(response)
        responseObj = response.json()

        if (responseObj == {"user":"updated."}):
            print("PERFIL ACTUALIZADO")
        else:
            print("USUARIO/CONTRASEÑA INCORRECTA")

        user = "correct";
    if option=='2':

        print("Nombre de usuario: ")
        usuario = input()
        print("Contraseña: ")
        password = input()
        print("Contraseña nueva: ")
        new_password = input()

        response = requests.put("https://127.0.0.1:5000/users", json={"option":option,"newpass":new_password,"name":usuario,"email":password}, verify='cert.pem')
        print(response)
        responseObj = response.json()

        if (responseObj == {"user":"updated."}):
            print("PERFIL ACTUALIZADO")
        else:
            print("USUARIO/CONTRASEÑA INCORRECTA")

        user = "correct";
ept:
 print("Error al conectar con el servidor de Registro")
```

Para editar usuario, dependiendo de la opción, ya sea editar usuario o contraseña, usaremos requests.put con los parámetros IP:PUERTO, un json con la opción, la nueva contraseña, y los datos de logueo del usuario y por último la clave pública para conectarse mediante HTTPS al servidor API.

```
    while contra != password:
        print("Contraseña: ")
        password = input()
        print("Repetir contraseña: ")
        contra = input()
        if(contra != password):
            print("Las contraseñas no coinciden")


    response = requests.post("https://127.0.0.1:5000/users", json={"name":usuario,"email":password}, verify='cert.pem')
    print(response)
    responseObj = response.json()

    if(responseObj == {"user":"registered."}):
        print("USUARIO CREADO")
    else:
        print("USUARIO YA EXISTE")
```

Para registrar un usuario haremos requests.post y pasamos el nombre de usuario y contraseña, si no hay error, se registra en caso contrario trataremos el error ya sea por usuario ya existente o error por parte del servidor.

d. FWQ_Sensor

```python
# DEFAULT
users = 0
topic = "sensorinfo"
sensorId = -1

# Funcion que calcula cuantos usuarios hay
def sensorUsers():
    global users
    while(1):
        try:
            res = input()
            if int(res) <= 0:
                # gen random number
                users = random.randint(0, 200)
            else:
                users = int(res)

            print("[USER MANAGER] There is/are " + str(users) + " waiting right now!")
        except:
            print("[ERROR] Type a natural number")

# Funcion que envia los datos al gestor de Colas
def sendInfo(producer):
    while(1):
        send = str(sensorId) + ":" + str(users)
        print("[INFO] Sending the info: " + send + " to the topic: " + topic)
        producer.send(topic, send.encode('utf-8'))
        time1 = random.randint(1, 3)
        time.sleep(time1)
```

En cuanto al sensor, tenemos **dos funciones**, una que calcula **la cantidad de usuarios que hay** y el otro que **envía los datos al gestor** de colas mediante el topic de “sensorinfo”

e. FWQ_WaitingTimeServer.py

Aquí vamos a calcular los tiempos de espera, **actuando de consumidor del topic "sensorinfo"** anteriormente mencionado.
Decodificamos la información que consumimos y basada en esa información realizaremos **una estimación del tiempo de espera.**

```
# Calculadora de tiempos
def calculateTimes(attractions):
    global et
    consumer = KafkaConsumer('sensorinfo',bootstrap_servers='localhost:9092')
    for message in consumer:
        valor = message.value.decode(FORMAT)
        valor = valor.split(":")
        atId = int(valor[0])
        atValue = int(valor[1])
        if atId <= len(et):
            estimatedTime = int(atValue/attractions[atId-1][2])*attractions[atId-1][1]
            et[atId-1] = estimatedTime
            updatedTimes[atId-1] = datetime.now()
            print("[TIMES] The attraction " + str(atId) + " has a estimated time of " + str(int(atValue/attractions[atId-1][2])*attractions[atId-1][1]) + " mins")
```

Como **hemos implementado anteriormente**, tenemos una función **keepAlive()** que averigua si el sensor está en línea basándose en **timestamps.**

```
# Keep Alive con los sensores
def keepAlive():
    global updatedTimes
    global et
    while(1):
        for i in range(len(updatedTimes)):
            if(et[i] != -1):
                date1 = updatedTimes[i]
                dateNow = datetime.now()
                dif = (dateNow-date1).total_seconds()
                if dif >= 10:
                    print("[KEEP ALIVE] Connection lost with sensor id = " + str(i+1))
                    et[i] = -1
        time.sleep(5)
```

Por último, la función que se encarga de **enviar la información del servidor al cliente,** mientras tengamos conexión, **iremos enviando la información de tiempos.** Aquí recibiremos comandos los cuales pueden ser, **“info”, “exit” o “keepalive”,** dependiendo del comando, el servidor devolverá una información u otra.

```python
# Send info
def sendInfo(conn, addr):
    print(f"[SERVER] {addr} connected.")
    global et

    connected = True
    while connected:
        try:
            msg_length = conn.recv(HEADER).decode(FORMAT)
            if msg_length:
                msg_length = int(msg_length)
                msg = conn.recv(msg_length).decode(FORMAT)
                if msg == "info":
                    msg_send = ""
                    for i in range(len(et)):
                        msg_send = msg_send + str(et[i])
                        if i != len(et)-1:
                            msg_send = msg_send + ":"
                    conn.send(f"{msg_send}".encode(FORMAT))
                    print("[SERVER] Sending the information!")

                elif msg == "exit":
                    connected = False
                elif msg == "keepalive":
                    conn.send(f"1".encode(FORMAT))
                else:
                    conn.send(f"Please type info to get all the estimated times or exit!".encode(FORMAT))
        except:
            print("[SERVER] Connection closed by the client :(")
            connected = False

    print("[SERVER] Disconnected user")
    conn.close()
```

f. API_REST

Tenemos varias funciones de tipo GET, PUT, POST
Donde se ejecuta el servidor, la funcion app.run tiene por parametro las claves publicas y privadas para realizar la conexión por HTTPS

```
# run
if __name__ == '__main__':
    app.run(host="127.0.0.1", port=5000,ssl_context=('cert.pem', 'key.pem'))
```

Estas claves las hemos creado mediante openssl siendo cert.pem la clave pública que usará el visitante para conectarse.

REGISTRO

```
# Crear usuario
    cursor.execute('''insert into users(username,password) values (?,?);''',(request.json['name'], generate_password_hash(request.json['email'])))
    sqliteConnection.commit()
    print("USUARIO CREADO")
    cursor.execute('''insert into auditoria(DATE,USUARIO,IP,ACCION_USER,DESC_USER) values (?,?,?,?,?);''',(date,request.json['name'],str(ip_address),"ALTA","usuario dado de alta"))
    sqliteConnection.commit()
    # AUDITORIA DE USUARIO, CREAR STRING PONER EN LOGS.txt
    user_log = str(date)+" | "+request.json['name']+" -- "+str(ip_address)+" | ALTA | usuario dado de alta | "
    text_file.write(user_log+'\n')
    text_file.close()


    cursor.close()
except sqlite3.Error as error:
    print("USUARIO YA EXISTE")


    cursor.execute('''insert into auditoria(DATE,USUARIO,IP,ACCION_USER,DESC_USER) values (?,?,?,?,?);''',(date,request.json['name'],str(ip_address),"ERROR","usuario ya existe"))
    sqliteConnection.commit()
    cursor.close()
    # AUDITORIA DE USUARIO, CREAR STRING PONER EN LOGS.txt
    user_log = str(date)+" | "+request.json['name']+" -- "+str(ip_address)+" | ERROR | USUARIO YA EXISTE | "
    text_file.write(user_log+'\n')
    text_file.close()

    return jsonify({'user': "exists."}), 400
finally:
    if sqliteConnection:
        sqliteConnection.close()
        print("The SQLite connection is closed")


return jsonify({'user': "registered."}), 201
```

El registro del usuario se realiza aquí, tras comprobar que los parámetros pasados son los correctos (Si no, devuelve código 400 y almacena el error el la base de datos) nos conectamos a la base de datos y insertamos los datos pasados, si el Insert devuelve error, es que el usuario ya existe, en caso contrario devolvemos 200. Todo este proceso se guarda haciendo auditoría de lo que pasa, ésto se guarda tanto en la base de datos como en un .txt llamado LOGS. Las contraseñas se cifran mediante un algoritmo de cifrado.
En concreto hemos usado el algoritmo de Werkzeug para el cifrado de contraseñas.

Editación de usuario / nombre de usuario

```python
# Editar usuario
try:
    userNew = [request.json['newname'], request.json['name'], request.json['email']]
    print(userNew[1])
    cursor.execute("SELECT password FROM users WHERE username = ?", (userNew[1],))
    sqliteConnection.commit()
    data = cursor.fetchall()

    if data==[]:
        raise sqlite3.Error()

    if check_password_hash(data[0][0], userNew[2]):
        print("USUARIO LOGGEADO")
        cursor.execute("UPDATE users SET username = ? WHERE username = ?", (userNew[0], userNew[1],))
        sqliteConnection.commit()
        print("USUARIO ACTUALIZADO")
        cursor.execute('''insert into auditoria(DATE,USUARIO,IP,ACCION_USER,DESC_USER) values (?,?,?,?,?);''',(date,request.json['name'],str(ip_address),"MODIFICACIÓN","usuario ha modificado nombre de usuario"))
        sqliteConnection.commit()
        # AUDITORIA DE USUARIO, CREAR STRING PONER EN LOGS.txt
        user_log = str(date)+" | "+userNew[0]+" -- "+str(ip_address)+" | MODIFICACION | Usuario ha modificado su nombre de usuario | "
        text_file.write(user_log+'\n')
        text_file.close()
    else:
        raise sqlite3.Error()

    cursor.close()
except sqlite3.Error as error:
    print("USUARIO/CONTRASEÑA INCORRECTA")


    cursor.execute('''insert into auditoria(DATE,USUARIO,IP,ACCION_USER,DESC_USER) values (?,?,?,?,?);''',(date,request.json['name'],str(ip_address),"ERROR","usuario usuario/contraseña incorrecta"))
    sqliteConnection.commit()
    cursor.close()
    # AUDITORIA DE USUARIO, CREAR STRING PONER EN LOGS.txt
    user_log = str(date)+" | "+userNew[0]+" -- "+str(ip_address)+" | ERROR | USUARIO/CONTRASEÑA INCORRECTA | "
    text_file.write(user_log+'\n')
    text_file.close()
    return jsonify({'user': "incorrect."}), 404
finally:
    if sqliteConnection:
        sqliteConnection.close()
        print("The SQLite connection is closed")
```

Aquí logueamos el usuario comprobando que los datos pasados coinciden con los de la base de datos, si es el caso, haremos UPDATE del nombre de usuario con el que se pasa por parámetro. Si en cualquier caso hay un error, éste se trata y devuelve 404 o 400.

Editación de contraseña

```python
if check_password_hash(data[0][0], userNew[2]):
    print("USUARIO LOGGEADO")
    cursor.execute("UPDATE users SET password = ? WHERE username = ?", (generate_password_hash(userNew[0]), userNew[1],))
    sqliteConnection.commit()
    print("USUARIO ACTUALIZADO")
    cursor.execute('''insert into auditoria(DATE,USUARIO,IP,ACCION_USER,DESC_USER) values (?,?,?,?,?);''',(date,request.json['name'],str(ip_address),"MODIFICACIÓN","usuario ha modificado su contraseña"))
    sqliteConnection.commit()
    # AUDITORIA DE USUARIO, CREAR STRING PONER EN LOGS.txt
    user_log = str(date)+" | "+userNew[0]+" -- "+str(ip_address)+" | MODIFICACION | Usuario ha modificado su contraseña | "
    text_file.write(user_log+'\n')
    text_file.close()
else:
    raise sqlite3.Error()
```

Aquí logueamos el usuario comprobando que los datos pasados coinciden con los de la base de datos, si es el caso, haremos UPDATE de la contraseña con el que se pasa por parámetro. Si en cualquier caso hay un error, éste se trata y devuelve 404 o 400.

g. API_ENGINE

Todos los métodos implementados son de tipo ‘GET’ puesto que solo necesitamos obtener información de otros módulos.

```python
@app.route('/map', methods=['GET'])
def getAll():
    conDb = sqlite3.connect(RUTE_DB)
    cur = conDb.cursor()

    cur =  cur.execute('SELECT * FROM map_info')
    row = cur.fetchone()
    tiempo = int(row[0])
    map = {}

    if(time.time()-tiempo > 10):
        map["online"] = 0
    else:
        map["online"] = 1


    e = list()
    for row in cur.execute('SELECT * FROM map_attractions'
        x = {
            "id": row[0],
            "X": row[1],
            "Y": row[2],
            "et": row[3],
            "cuadrante": row[4]
        }
        e.append(x)

    map["attractions"] = e

    e = list()
    for row in cur.execute('SELECT * FROM map_users'):
        x = {
            "id": row[0],
            "at": row[1],
            "X": row[2],
            "Y": row[3],
            "nombre": row[4]
        }
        e.append(x)

    map["users"] = e

    e = list()
    for row in cur.execute('SELECT * FROM map_cities'):
        x = {
            "id": row[0],
            "temp": row[1],
        }
        e.append(x)

    map["cities"] = e

    return jsonify(map), 200
```

Este método lo que hace es obtener toda la información relevante al mapa con la extensión /map.

Se realiza un select a la base de datos donde se obtiene las atracciones, los usuarios y las ciudades pertenecientes al mapa

```python
@app.route('/map/users', methods=['GET'])
def getUsers():
    conDb = sqlite3.connect(RUTE_DB)
    cur = conDb.cursor()
    users = {
        "users": []
    }
    e = list()
    for row in cur.execute('SELECT * FROM map_users'):
        x = {
            "id": row[0],
            "at": row[1],
            "X": row[2],
            "Y": row[3],
            "nombre": row[4]
        }
        e.append(x)

    users["users"] = e
    conDb.close()
    return(jsonify(users)), 200

@app.route('/map/cities', methods=['GET'])
def getCities():
    conDb = sqlite3.connect(RUTE_DB)
    cur = conDb.cursor()
    cities = {
        "cities": []
    }
    e = list()
    for row in cur.execute('SELECT * FROM map_cities'):
        x = {
            "id": row[0],
            "temp": row[1],
        }
        e.append(x)

    cities["cities"] = e
    conDb.close()
    return(jsonify(cities)), 200

@app.route('/map/info', methods=['GET'])
def getInfo():
    conDb = sqlite3.connect(RUTE_DB)
    cur = conDb.cursor()

    cur =  cur.execute('SELECT * FROM map_info')
    row = cur.fetchone()
    tiempo = int(row[0])
    conDb.close()

    if(time.time()-tiempo > 10):
        return jsonify({"online": 0}), 200
    else:
        return jsonify({"online": 1}), 200
```

Aquí tenemos varias funciones que básicamente sirven de ‘Getters’ para información perteneciente a la base de datos.

GetUsers obtiene los usuarios con la extensión /map/users

GetCities obtiene las ciudades con la extensión /map/cities

GetInfo obtiene información relevante al mapa con la extensión /map/info

```python
# REGISTRO
@app.route('/map/attractions', methods=['GET'])
def getAttractions():
    conDb = sqlite3.connect(RUTE_DB)
    cur = conDb.cursor()
    attractions = {
        "attractions": []
    }
    e = list()
    for row in cur.execute('SELECT * FROM map_attractions'):
        x = {
            "id": row[0],
            "X": row[1],
            "Y": row[2],
            "et": row[3],
            "cuadrante": row[4]
        }
        e.append(x)

    attractions["attractions"] = e
    conDb.close()
    return(jsonify(attractions)), 200
```

GetAtractions obtiene las atracciones con la extensión /map/attractions

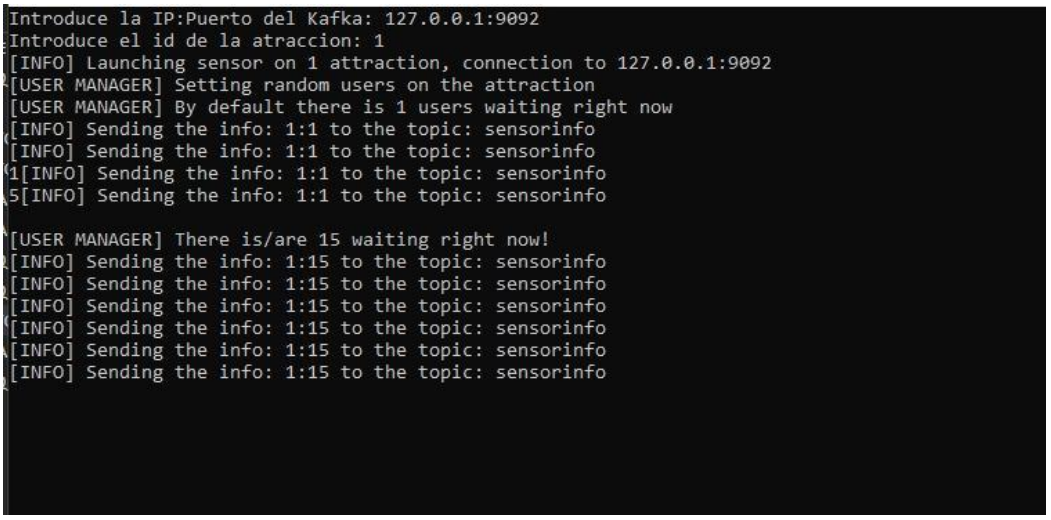
FERNET

Hemos implementado el uso de fernet para el cifrado simétrico de mensajes.

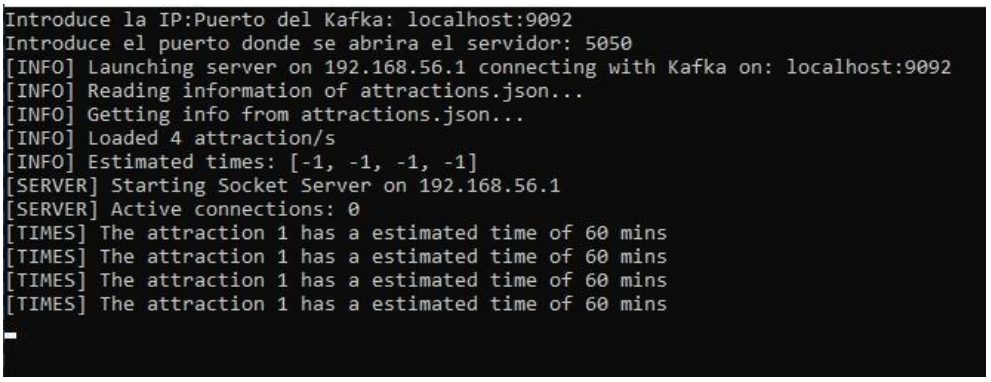
```python
# Importamos Fernet
from cryptography.fernet import Fernet

# Generamos una clave
clave = Fernet.generate_key()
clave2 = Fernet.generate_key()
print(clave)
# Creamos la instancia de Fernet
# Parametros: key: clave generada
f = Fernet("hzfiGUcAugyyAHhRzOORPxNXyG_VNmupYO8oWDvLd8A=")

# Encriptamos el mensaje
# utilizando el método "encrypt"
token = f.encrypt(b'Mensaje secreto')

# Mostramos el token del mensaje
print(token)

# Podemos descifrar el mensaje utilizando
# el método "decrypt".
des = f.decrypt("gAAAAABhykNb_yDbYjtJs8QFpKuiyW8RenDCVjdIf2F8HNGdEEwe05u-ymvX-0Te5L5grgtsI01QJD2NZ3vRdq
print(des.decode())
```

2. Capturas de pantalla mostrando el funcionamiento

FWQ_Sensor.py



FWQ_WaitingTimeServer.py



FWQ_Engine.py

```
Introduce la IP:Puerto del Kafka: localhost:9092
Introduce el numero de visitiantes maximos: 10
Introduce la ip:puerto del servidor de tiempos de espera:  192.168.56.1:5050
[DATABASE] Connecting to data base
[DATABASE] Getting information of attractions
[DATABASE] There is/are 4 attractions
[MODULE] Starting Waiting Time Server module...
[WAITING TIME SERVER] Starting task
[MODULE] Starting Login System
[MODULE] Starting Kafka Consumer for User Info
[WAITING TIME SERVER] Connecting on ('192.168.56.1', 5050)
[MODULE] Starting Map Sender
[MODULE] Starting Keep Alive User
[WAITING TIME SERVER] Requesting info...
[WAITING TIME SERVER] Got a response: 60:-1:-1:-1
[LOGIN] Awaiting for info on Kafka Server topic = logindetails
[MAP SENDER] Sending map to the topic: mapinfo
[WAITING TIME SERVER] Requesting info...
[WAITING TIME SERVER] Got a response: 60:-1:-1:-1
[MAP SENDER] Sending map to the topic: mapinfo
[WAITING TIME SERVER] Requesting info...
[WAITING TIME SERVER] Got a response: 60:-1:-1:-1
[WAITING TIME SERVER] Requesting info...
[WAITING TIME SERVER] Got a response: 60:-1:-1:-1
```

FWQ_Visitors.py

```
Introduce la IP:Puerto del Kafka: localhost:9092
Introduce la IP:Puerto del Registry: 192.168.56.1:8100
##############################################################################################################
## BIENVENIDO AL RESORT                                                                                     ##
##                                                                                                          ##
## Menu:                                                                                                    ##
## 1. Entrar al parque                                                                                      ##
## 2. Editar perfil                                                                                         ##
## 3. Registrarse                                                                                           ##
## 4. Salir                                                                                                 ##
##############################################################################################################
Introduce la opcion que quieras hacer:
1
Introduce tu nombre de usuario:
sergio
Introduce tu contraseña:
sergio
Estableciendo conexion con el parque...
```

```
      1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16   17   18   19   20
================================================================================================================
1|    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
2|    -    -    -    ?    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
3|    -    -    -    -    -    -    -    ?    -    -    -    -    -    -    -    -    -    -    -    -
4|    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
5|    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
6|    -    -    -    -    ?    60   -    -    YO   -    -    -    -    -    -    -    -    -    -    -
7|    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
8|    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
9|    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
10|   -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
11|   -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
12|   -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
13|   -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
14|   -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
15|   -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
16|   -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
17|   -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
18|   -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
19|   -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
20|   -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -

USUARIO         POSICION ACTUAL         DESTINO
YO              [9, 6]                  [6, 6]
```

```
##############################################################################################################
## BIENVENIDO AL RESORT                                                                                     ##
##                                                                                                          ##
## Menu:                                                                                                    ##
## 1. Entrar al parque                                                                                      ##
## 2. Editar perfil                                                                                         ##
## 3. Registrarse                                                                                           ##
## 4. Salir                                                                                                 ##
##############################################################################################################
Introduce la opcion que quieras hacer:
3
Establecida conexión en [('192.168.56.1', 1000)]
Nombre de usuario:
kafka
Contraseña:
kafka
Repetir contraseña:
kafka
USUARIO CREADO
##############################################################################################################
```

FRONT

1	#	#	#	#	#	#	#	#	#	#	-	-	-	-	-	-	-	-	-	-
2	#	#	#	[2/?]	#	#	#	#	#	#	-	-	-	-	-	-	-	-	-	-
3	#	#	#	#	#	#	#	[3/?]	#	#	-	-	-	-	-	-	-	[14/?]	-	-
4	#	#	#	#	#	#	#	#	#	#	-	-	-	-	-	-	-	-	-	-
5	#	#	#	#	#	#	#	#	#	#	-	-	-	[12/?]	-	-	-	-	-	-
6	#	#	#	#	[4/?]	[1/?]	#	#	[9/?]	#	-	-	-	-	-	-	-	-	-	-
7	#	#	#	#	#	#	#	#	#	#	-	-	-	-	-	-	-	-	-	-
8	#	#	#	#	#	#	#	#	#	#	-	-	-	-	-	-	-	-	-	-
9	#	#	#	#	#	#	#	#	#	#	-	-	-	-	-	-	-	-	-	-
10	#	#	#	[6/?]	#	#	#	#	#	#	[10/?]	-	-	-	-	-	-	-	-	-
11	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
12	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	[13/?]	X	X	X	X
13	X	X	X	X	X	X	X	[8/?]	X	X	X	X	X	X	X	X	X	X	X	X
14	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
15	X	X	[5/?]	X	X	X	X	X	X	X	X	[11/?]	X	X	X	X	X	X	X	X
16	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
17	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
18	X	X	X	X	X	[7/?]	X	X	X	X	X	X	X	X	X	X	X	X	X	X
19	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	[15/?]	X
20	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Visitantes
No hay usuarios dentro del parque en este momento
Temperatura
CUADRANTE ARRIBA IZQUIERDA: ¡¡No se conoce el tiempo!!
CUADRANTE ARRIBA DERECHA: Zona abierta, clima estable, pasatelo bien :) -> 24ºC
CUADRANTE ABAJO IZQUIERDA: ¡¡Zona Cerrada!! La temperatura no es apta -> 16ºC
CUADRANTE ABAJO DERECHA: ¡¡Zona Cerrada!! La temperatura no es apta -> 9ºC

FunWithQueues Inicio

No se ha podido conectar al servidor API

Leyenda del mapa

Estado del parque
[#] = No se conoce información sobre la temperatura en esta zona
[X] = La zona del parque está cerrada
[-] = La zona del parque se encuentra activa

Visitantes
[nombre] = Los visitantes que se encuentren en el mapa tendrán un pseudonombre, sus 2 caracteres de su usuario, en la posición donde se encuentren

Atracciones
[id/Tiempo] = Si no se conoce el tiempo de una atraccion se mostrará ? [id/?]

Visitantes
Visitantes
Temperatura

© Fun With Queues

API VISITANTE

```
##########################################################################################################
## BIENVENIDO AL RESORT                                                                                  ##
##                                                                                                       ##
## Menu:                                                                                                 ##
## 1. Entrar al parque                                                                                   ##
## 2. Editar perfil (EDIT VIA API)                                                                       ##
## 3. Editar perfil (EDIT VIA SOCKET)                                                                    ##
## 4. Registrarse (REGISTER VIA API)                                                                     ##
## 5. Registrarse (REGISTER VIA SOCKET)                                                                  ##
## 6. Salir                                                                                              ##
##########################################################################################################
Introduce la opcion que quieras hacer:
4
Nombre de usuario:
pepe
Contraseña:
contraPEPE
Repetir contraseña:
contraPEPE
<Response [201]>
USUARIO CREADO
```

API_REST

```
 * Serving Flask app 'API_Rest' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on https://127.0.0.1:5000/ (Press CTRL+C to quit)
Successfully Connected to SQLite
USUARIO CREADO
The SQLite connection is closed
127.0.0.1 - - [29/Dec/2021 19:18:53] "POST /users HTTP/1.1" 201 -
```