# Diseño de la Lógica de Negocio del Sistema de Monitoreo Ambiental

Hecho por Santiago Aguirre

El sistema permite que ciudadanos o usuarios registrados midan niveles de contaminación atmosférica (como O3, CO2, NO2) utilizando un sensor Arduino con iBeacon. Los datos obtenidos se envían al móvil y luego a una base de datos, donde se almacenan junto con la información geográfica de la medición. Además, los usuarios pueden acumular puntos, registrar su actividad y visualizar estadísticas a través de una aplicación o página web.

El siguiente documento describe el diseño lógico de las funciones principales del sistema, así como su algoritmo y funcionamiento.

## Funciones Principales

### 1. registerUser(username, email, password, townHallId)

Registra un nuevo usuario en el sistema. Crea un registro en la tabla users con sus datos básicos, vinculándolo al ayuntamiento correspondiente y asignándole un rol por defecto. Además, inicializa los valores de actividad en 0.

| | Descripción |
|---|---|
| Objetivo | Crear un usuario nuevo y vincularlo con su ayuntamiento. |
| Entradas | username, email, password, townHallId |
| Salidas | Confirmación del registro o error (usuario duplicado). |
| Tablas usadas | users, roles, town_halls |

**Algoritmo (pseudocódigo)**

```
registerUser(username, email, password, townHallId)
      SI existeUsuarioCon(email) O existeUsuarioCon(username)
            RETORNAR "Error: Usuario o email ya existe"
      rolDefault ← buscarRol("walker")
      usuarioNuevo ← {username, email, cifrar(password),
```

```
    rolDefault.id, 0 puntos, 0 horas, 0 distancia, townHallId}
    INSERTAR usuarioNuevo EN users
    RETORNAR "Usuario registrado correctamente"
```

## 2. loginUser(email, password)

Verifica las credenciales de un usuario para permitir el acceso al sistema. Compara la contraseña ingresada con el hash almacenado en la base de datos. Si las credenciales son correctas, se genera un token de sesión.

| | Descripción |
|---|---|
| Objetivo | Autenticar a un usuario y crear sesión. |
| Entradas | username, password |
| Salidas | Token de sesión o mensaje de error. |
| Tablas usadas | users |

**Algoritmo (pseudocódigo)**

```
loginUser(username, password)
      usuario ← buscarUsuarioPorUsername(usernmae)
      SI usuario = NULL
            RETORNAR "Error: Usuario no encontrado"
      SI verificarPassword(password, usuario.password) = FALSO
      RETORNAR "Error: Contraseña incorrecta"
      tokenSesion ← generarToken(usuario.id)
         RETORNAR tokenSesion
```

## 3. getUser(userId)

Obtiene la información básica de un usuario registrado. Esta función no maneja contraseñas ni procesos de autenticación; su objetivo es mostrar los datos del perfil, puntos y actividad.

| | Descripción |
|---|---|
| Objetivo | Consultar el perfil de un usuario. |
| Entradas | userId |

| | |
|---|---|
| Salidas | Datos del usuario: nombre, correo, puntos, tiempo total, distancia total, rol, ayuntamiento. |
| Tablas usadas | users, town_halls, roles |

**Algoritmo (pseudocódigo)**

```
getUser(userId)
	usuario ← buscarUsuarioPorId(userId)
	SI usuario = NULL
		RETORNAR "Error: Usuario no encontrado"
	perfil ← {username, email, points, active_hours, total_distance,
	town_hall, role}
	RETORNAR perfil
```

# 4. linkNodeToUser(userId, nodeName)

Vincula un nodo físico (por ejemplo, un sensor Arduino con iBeacon) a un usuario. Esto permite asociar las mediciones tomadas por el dispositivo con una cuenta específica.

| | Descripción |
|---|---|
| Objetivo | Registrar un nuevo nodo asociado a un usuario. |
| Entradas | userId, nodeName |
| Salidas | Confirmación de vinculación o error si el usuario no existe. |
| Tablas usadas | nodes, users |

**Algoritmo (pseudocódigo)**

```
linkNodeToUser(userId, nodeName)
	usuario ← buscarUsuarioPorId(userId)
	SI usuario = NULL
		RETORNAR "Error: Usuario no encontrado"
	nodoNuevo ← {user_id, nodeName, status='active',
	lastStatusUpdate=tiempoActual()}
	INSERTAR nodoNuevo EN nodes
	RETORNAR "Nodo vinculado correctamente"
```

## 5. updateUserActivity(userId, time, distance)

Actualiza los datos de actividad física del usuario, sumando las horas activas y la distancia total recorrida. Este proceso puede ejecutarse cada vez que se detecta un movimiento o una medición realizada desde una ubicación diferente.

| | Descripción |
|---|---|
| Objetivo | Incrementar tiempo y distancia del usuario. |
| Entradas | userId, time, distance |
| Salidas | Confirmación de actualización o error si el usuario no existe. |
| Tablas usadas | users |

**Algoritmo (pseudocódigo)**

```
updateUserActivity(userId, time, distance)
      usuario ← buscarUsuarioPorId(userId)
      SI usuario = NULL
            RETORNAR "Error: Usuario no encontrado"
      usuario.active_hours ← usuario.active_hours + time
      usuario.total_distance ← usuario.total_distance + distance
      ACTUALIZAR users SET
            active_hours, total_distance
      DONDE id = userId
      RETORNAR "Actividad actualizada correctamente"
```

# Resumen general del flujo de negocio

1. El usuario se registra mediante registerUser().
2. El usuario inicia sesión con loginUser().
3. Tras iniciar sesión, puede consultar su información con getUser().
4. Si tiene un sensor físico, se vincula a su cuenta con linkNodeToUser().
5. Conforme se mueva o tome mediciones, el sistema actualiza su actividad con updateUserActivity().

Este conjunto de funciones conforma la base de la lógica de negocio del sistema, permitiendo el registro, autenticación, gestión de nodos y seguimiento de la actividad del usuario.