**CONTROL DEL DOCUMENTO**

Registro de Cambios

|  |  |  |  |
| --- | --- | --- | --- |
| **Fecha** | **Autor** | **Versión** | **Referencia de Cambios** |
|  |  |  |  |
| 26/11/2019 | Juan Carlos Roeder | 1.0.0 | Creación de documentación |
| 27/11/2019 | Sheyla Pisfil Ballesteros | 1.0.1 | Añadió requerimiento |
| 02/11/2019 | Sheyla Pisfil Ballesteros | 1.0.2 | Añadió requerimiento |
|  |  |  |  |

Revisiones

|  |  |
| --- | --- |
| **Fecha** | **Nombre** |
|  |  |
|  |  |
|  |  |
|  |  |

Distribución

|  |  |  |
| --- | --- | --- |
| **N°. Copia** | **Nombre** | **Ubicación** |
|  |  |  |
|  |  |  |
|  |  |  |

**TABLA DE CONTENIDO**

**1.** [**SOLUCIÓN**](#_484rgehfw0gj) **3**

1.1. [Objetivo de la solución](#_1fob9te) 3

1.2. [Resumen de la solución](#_3znysh7) 3

1.3. [Contexto Funcional [Opcional]](#_2et92p0) 3

1.4. [Definiciones, acrónimos y Abreviaciones [Opcional]](#_tyjcwt) 3

**2.** [**AMBIENTE OPERATIVO**](#_wa4vzbiw0d3t) **4**

[2.1. Sistema Operativo:](#_1t3h5sf) 4

[2.2. Servidor Web:](#_ge9laxi609ej) 4

[2.3. Servidor de base de datos](#_rwtdrzbk0t8) 4

[2.4. Lenguaje de programación del servidor](#_mddj36fcor8v) 4

[2.5. Lenguaje de programación del Móvil](#_qfcmadab78m8) 4

**3.** [**REQUERIMIENTOS FUNCIONALES**](#_xsokde95bj0e) **5**

3.1. [Listado de Requerimientos Funcionales.](#_35nkun2) 6

3.2. [Solución Móvil.](#_hfbnqbu83302) 7

3.3. [Dependencias de Solución Móvil.](#_t2qg8mwq1ofc) 8

3.4. [Solución web.](#_c6w3kb87hp0c) 9

3.5. [Dependencias de la Solución Web.](#_qjcg96xxxac7) 10

# SOLUCIÓN

## Objetivo de la solución

[Se ingresa el objetivo y/o justificación de la solución a nivel de negocio.]

## Resumen de la solución

[Se ingresa el resumen de la solución a nivel funcional]

## Contexto Funcional [Opcional]

[Dependiendo de la complejidad de la solución se ingresan lineamientos, proceso, procedimientos, etc.]

## Definiciones, acrónimos y Abreviaciones [Opcional]

[Dependiendo de la solución se ingresa las definiciones, acrónimos y/o abreviaciones pertinentes para que se entienda el análisis de la solución.]

# AMBIENTE OPERATIVO

## 2.1 Sistema Operativo:

* WINDOWS SERVER 2012/R2 STANDARD 64 BITS

## 2.2 Servidor Web:

* INTERNET INFORMATION SERVICES 8
* DTSX VERSION=11.0.0.0

## 2.3 Servidor de base de datos

* SQL SERVER 2012/R2 SP3
* SQL SERVER INTEGRATION SERVICES 2012/R2

## 2.4 Lenguaje de programación del servidor

* .NET FRAMEWORK 4
* HTML5
* CSS

## 2.5 Lenguaje de programación del Móvil

* ANDROID

# REQUERIMIENTOS FUNCIONALES

En esta sección se detalla los aspectos esenciales del software a construir.

**Dónde:**

Para los requerimientos funcionales las prioridades servirán para identificar lo relevante de la funcionalidad y para enfocar las pruebas más exhaustivamente donde se identifique.

**Prioridad:**

Crítico **(C)**, Necesario **(N)**, Deseable **(D)**

## Listado de Requerimientos Funcionales.

|  |  |  |
| --- | --- | --- |
| **Nro** | **CÓDIGO REQ FUNCIONAL** | **NOMBRE REQUERIMIENTO FUNCIONAL** |
| [01](#kagjpyi7v748) | [REQ 01](#kagjpyi7v748) | MICROSERVICIOS - API Management Usuario |
| [02](#vnl8b6nzchnn) | [REQ 02](#vnl8b6nzchnn) | [MICROSERVICIOS - API Management Actividad](#vnl8b6nzchnn) |
| [03](#pp4cides6rju) | [REQ 03](#pp4cides6rju) | [MICROSERVICIOS - API Management Alerta](#pp4cides6rju) |
| [04](#3irm12wpc390) | [REQ 04](#3irm12wpc390) | [MICROSERVICIOS](#3irm12wpc390) - API Management Foto |
| 05 | [REQ 05](#3irm12wpc390) | [MICROSERVICIOS](#3irm12wpc390) - API Management Riesgo |
| 06 | [REQ 06](#3irm12wpc390) | [MICROSERVICIOS](#3irm12wpc390) - API Management Empresa |
| [07](#3irm12wpc390) | [REQ 07](#3irm12wpc390) | [MICROSERVICIOS](#3irm12wpc390) - API Management Site |
| [08](#3irm12wpc390) | [REQ 08](#3irm12wpc390) | [MICROSERVICIOS](#3irm12wpc390) - API Management Nivel |
| [09](#3irm12wpc390) | [REQ 09](#3irm12wpc390) | [MICROSERVICIOS](#3irm12wpc390) - API Management Rol |
| 10 | [REQ 10](#3irm12wpc390) | [MICROSERVICIOS](#3irm12wpc390) - API Management Token |

## Solución Microservicios.

Elementos en común de todos los microservicios:

RESPONSE ERROR 400:

{

"result": {

"code": 400,

"description": "Request incorrecto",

"descriptionDetail": "Request incorrecto",

"timeStamp": "2020-04-26T16:56:59.000",

"transactionId": 125312647124

}

}

RESPONSE ERROR 401:

{

"result": {

"code": 401,

"description": "Acceso Denegado",

"descriptionDetail": "Token inválido",

"timeStamp": "2020-04-26T16:56:59.000",

"transactionId": 125312647124

}

}

RESPONSE ERROR 404:

{

"result": {

"code": 404,

"description": "Recurso no encontrado",

"descriptionDetail": " Recurso no encontrado ",

"timeStamp": "2020-04-26T16:56:59.000",

"transactionId": 125312647124

}

}

RESPONSE ERROR 500:

{

"result": {

"code": 500,

"description": "Error interno en el servidor",

"descriptionDetail": "Error en el procesamiento de la solicitud",

"timeStamp": "2020-04-26T16:56:59.000",

"transactionId": 125312647124

}

}

HEADER:

Content-Type: application/json

Authorization: Bearer {{token}}

applicationCode: APP-MOB

requestTimestamp: 2018-04-09T12:34:00.544-03:00

|  |  |
| --- | --- |
| **REQ # 01 - MICROSERVICIOS - API Management Usuario** | |
| Item: | 01 |
| Precondición | No aplica |
| Flujo normal | 1. Servicios: 2. Obtener usuarios, mediante filtros:   /api/listarUsuarios – POST  REQUEST:  {  "nombres": "Jhensson Frank",  "apellidos": "Ayma Flores",  "usuario": "jhensson2000",  "estado": 1  "numero\_pagina" 1,  "numero\_elementos": 20  }  COMENTARIOS DEL REQUEST: Si "nombre", "apellido", "usuario" o "estado" son enviados como campos vacíos, no serán considerados para el filtro. Y si envío el parámetro "numero\_pagina" como 0, no se realizará filtro de paginación.  RESPONSE OK:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Usuarios encontrados correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  },  "usuario": [  {  "nombres": "Jhensson Frank",  "apellidos": "Ayma Flores",  "usuario": "jhensson2000",  "correo": "jhensson.ayma@gmail.com",  "estado": 1,  "id\_rol": 1,  "id\_empresa": 1  }  ]  }   1. Registrar un usuario:   /api/usuario - POST  REQUEST:  {  "nombres": "Jhensson Frank",  "apellidos": "Ayma Flores",  "usuario": "jhensson2000",  "correo": "jhensson.ayma@gmail.com",  "contrasena": "cdd4f874095045f4ae6670038cbbd05fac9d4802",  "estado": 1,  "id\_rol": 1,  "id\_empresa": 1  }  RESPONSE OK:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Usuario registrado correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  }  }   1. Editar un usuario:   /api/usuario – PUT  REQUEST:  {  "id": 23  "nombres": "Jhensson Frank",  "apellidos": "Ayma Flores",  "correo": "jhensson.ayma@gmail.com",  "contrasena": "cdd4f874095045f4ae6670038cbbd05fac9d4802",  "estado": 1,  "id\_rol": 1,  "id\_empresa": 1  }  RESPONSE:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Usuario editado correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  }  }   1. Validar si un usuario existe en base de datos, a través de los campos usuario y password:   /api/usuario/validate – POST  REQUEST:  {  "usuario": "jhensson2000",  "contrasena": "cdd4f874095045f4ae6670038cbbd05fac9d4802"  }  RESPONSE OK:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Usuario validado correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  },  "usuario": {  "nombres": "Jhensson Frank",  "apellidos": "Ayma Flores",  "usuario": "jhensson2000",  "correo": "jhensson.ayma@gmail.com",  "estado": 1,  "id\_rol": 1,  "id\_empresa: 1  }  }   1. Tecnologías de desarrollo:  * Spring Boot 2.2 * Spring Data JPA 2.2  1. Base de datos: MySQL 8 |
| Prioridad | Crítico (**C**) |
| Validaciones | No aplica |
| Flujo alternativo | No aplica |
| Criterio de aceptación del usuario final | Los servicios de APIMUsuario deben devolver la información en formato Json, donde necesariamente se debe detallar una descripción y un código HTTP. |

|  |  |
| --- | --- |
| **REQ # 02 - MICROSERVICIOS - API Management Actividad** | |
| Item: | 02 |
| Precondición | No aplica |
| Flujo normal | 1. Servicios: 2. Obtener actividades, mediante filtros:   /api/listarActividades – POST  REQUEST:  {  "nombre": "Actividad 5",  "id\_empresa": 1,  "id\_nivel": 2,  "id\_riesgo": 3,  "fecha\_inicio": "2020-05-05 05:05:05",  "fecha\_fin": "2020-05-08 05:08:05",  "estado": 1  "numero\_pagina": 1,  "numero\_elementos": 20  }  COMENTARIOS DEL REQUEST: Si "nombre", "id\_empresa", "id\_nivel", "id\_riesgo", "fecha\_inicio", "fecha\_fin" o "estado" son enviados como campos vacíos, no serán considerados para el filtro. Y si envío el parámetro "numero\_pagina" como 0, no se realizará filtro de paginación.  RESPONSE OK:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Actividades encontradas correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  },  "actividad": [  {  "nombre": "Actividad 1",  "fecha": "2020-05-05 05:05:05",  "nro\_trabajadores": "5",  "observaciones": "Lorem ipsum ...",  "estado": "1",  "latitud": "13.23",  "longitud": "18.23",  "id\_site": "1",  "id\_nivel": "2",  "id\_empresa": "5",  "id\_operativo": "4",  "id\_sponsor": "2",  "riesgo": [  {  "nombres": "Riesgo 1",  "ruta\_foto": " https:awsS3/38trtgrngn/photo01.jpg",  "estado": "1"  }  ],  "historial": [  {  "latitud": "13.23",  "longitud": "18.23",  "motivo": "Tormenta",  "motivo\_by": "jhensson",  "observarciones": "Posible corto circuito"  }  ]  }  ]  }   1. Obtener el detalle de una sola actividad:   /api/actividad/{id} – GET  PARAMETERS:   * id: Identificador de la actividad   RESPONSE:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Actividad encontrada correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  },  "actividad": {  "nombre": "Actividad 1",  "fecha": "2020-05-05 05:05:05",  "nro\_trabajadores": "5",  "observaciones": "Hola",  "estado": "1",  "latitud": "13.23",  "longitud": "18.23",  "id\_site": "1",  "id\_nivel": "1",  "id\_empresa": "1",  "id\_operativo": "1",  "id\_sponsor": "1",  "riesgo": [  {  "nombres": "Riesgo 1",  "ruta\_foto": "https:awsS3/38trtgrngn/photo01.jpg",  "estado": "1"  }  ],  "historial": [  {  "latitud": "13.23",  "longitud": "18.23",  "motivo": "Tormenta",  "motivo\_by": "jhensson",  "observarciones": "Posible corto circuito"  }  ]  }  }   1. Registrar una actividad:   /api/actividad – POST  REQUEST:  {  "nombre": "Actividad 1",  "fecha": "2020-05-05 05:05:05",  "nro\_trabajadores": "5",  "observaciones": "Tormenta",  "descripción": "Actividad de alto riesgo",  "estado": "1",  "latitud": "13.23",  "longitud": "18.23",  "id\_site": "1",  "id\_nivel": "1",  "id\_empresa": "1",  "id\_operativo": "1",  "id\_sponsor": "1"  "riesgo": [  {  "id\_riesgo”: 1  "estado": "1"  }  ],  "historial": [  {  "latitud": "13.23",  "longitud": "18.23",  "motivo": "Tormenta",  "motivo\_by": "jhensson",  "observarciones": "Posible corto circuito"  }  ]  }  RESPONSE OK:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Actividad registrada correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  }  }   1. Editar una actividad:   /api/actividad – PUT  REQUEST:  {  "id\_actividad": 5,  "nombre": "Actividad 1",  "fecha": "2020-05-05 05:05:05",  "nro\_trabajadores": "5",  "observaciones": "Tormenta",  "descripción": "Actividad de alto riesgo",  "estado": "1",  "latitud": "13.23",  "longitud": "18.23",  "id\_site": "1",  "id\_nivel": "1",  "id\_empresa": "1",  "id\_operativo": "1",  "id\_sponsor": "1"  "riesgo": [  {  "id\_actividad\_riesgo": 23,  "nombres": "Riesgo 1",  "estado": "1"  }  ]  }  RESPONSE OK:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Actividad editada correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  }  }   1. Tecnologías de desarrollo:  * Spring Boot 2.2 * Spring Data JPA 2.2  1. Base de datos: MySQL 8 |
| Prioridad | Crítico (**C**) |
| Validaciones | No aplica |
| Flujo alternativo | No aplica |
| Criterio de aceptación del usuario final | Los servicios de APIMActividad deben devolver la información en formato Json, donde necesariamente se debe detallar una descripción y un código HTTP. |

|  |  |
| --- | --- |
| **REQ # 03 - MICROSERVICIOS - API Management Alerta** | |
| Item: | 03 |
| Precondición | No aplica |
| Flujo normal | 1. Servicios: 2. Obtener todas las alertas:   /api/actividad/{id}/listarAlertas - GET  PARAMETERS:  - id: Identificador de la actividad.  RESPONSE OK:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Alertas encontradas correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  },  “alerta”: [  {  "fecha": "2020-05-19 05:06:07",  "latitud": "43.23",  "longitud": "15.85",  "actividad": {  "nombre": "Actividad 1",  "site": "Sitio 1",  "supervisor": "Manuel Pérez",  "empresa": "DXC Technology",  "detalle": "Loren ipsum…"  }  }  ]  }   1. Registrar una alerta, además se debe enviar una notificación push utilizando Firebase:   /api/alerta – POST  REQUEST:  {  "fecha": "2020-05-19 05:06:07",  "latitud": "43.23",  "longitud": "15.85",  "id\_actividad": 22  }  RESPONSE OK:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Alerta registrada correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  }  }   1. Tecnologías de desarrollo:  * Spring Boot 2.2 * Spring Data JPA 2.2  1. Base de datos: MySQL 8 |
| Prioridad | Crítico (**C**) |
| Validaciones | No aplica |
| Flujo alternativo | No aplica |
| Criterio de aceptación del usuario final | Los servicios de APIMAlerta deben devolver la información en formato Json, donde necesariamente se debe detallar una descripción y un código HTTP. |

|  |  |
| --- | --- |
| **REQ # 04 - MICROSERVICIOS - API Management Foto** | |
| Item: | 04 |
| Precondición | No aplica |
| Flujo normal | 1. Services: 2. Obtener todas las fotos de una actividad:   /api/actividad/{id}/listarFotos - GET  PARAMETERS:  - id: Identificador de la actividad  RESPONSE OK:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Fotos encontradas correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  },  "foto": [  {  "ruta\_foto": "https:awsS3/38tf4rtgrngn/photo01.jpg",  "tipo": 1  }  ]  }   1. Registrar foto   /api/foto – POST  REQUEST:  {  "foto\_file": {imagen.jpg},  "tipo": 1  }  COMENTARIOS DEL REQUEST: El request será un HTTP multipart para permitir la transferencia de imágenes y datos al mismo tiempo al servidor. Además, se debe registrar la foto en el servicio S3 de AWS.  RESPONSE OK:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Foto registrada correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  }  }   1. Tecnologías de desarrollo:  * Spring Boot 2.2 * Spring Data JPA 2.2  1. Base de datos: MySQL 8 |
| Prioridad | Crítico (**C**) |
| Validaciones | No aplica |
| Flujo alternativo | No aplica |
| Criterio de aceptación del usuario final | Los servicios de APIMFoto deben devolver la información en formato Json, donde necesariamente se debe detallar una descripción y un código HTTP. |

|  |  |
| --- | --- |
| **REQ # 05 - MICROSERVICIOS - API Management Riesgo** | |
| Item: | 05 |
| Precondición | No aplica |
| Flujo normal | 1. Servicios: 2. Obtener todos los riesgos   /api/listarRiesgos – GET  PARAMETERS:  Ninguno  RESPONSE:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Riesgos encontrados correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  },  "riesgo": [  {  "nombre": "Riesgo 01",  "ruta\_foto": "https:awsS3/38tf4rtgrngn/photo01.png",  "estado": "1"  }  ]  }   1. Registrar un riesgo:   /api/riesgo – POST  REQUEST:  {  "nombre": "Riesgo 01",  "imagen\_file": {imagen.png},  "estado": "1"  }  COMENTARIOS DEL REQUEST: El request será un HTTP multipart para permitir la transferencia de imágenes y datos al mismo tiempo al servidor. Además, se debe registrar la foto en el servicio S3 de AWS.  RESPONSE:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Riesgo registrado correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  }  }   1. Editar un riesgo:   /api/riesgo - PUT  REQUEST:  {  "id": "4",  "nombre": "Riesgo 01",  "imagen\_file": {imagen.png},  "estado": "1"  }  COMENTARIOS DEL REQUEST: El request será un HTTP multipart para permitir la transferencia de imágenes y datos al mismo tiempo al servidor. Además, se debe editar la foto en el servicio S3 de AWS.  RESPONSE:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Riesgo editado correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  }  }   1. Tecnologías de desarrollo:  * Spring Boot 2.2 * Spring Data JPA 2.2  1. Base de datos: MySQL 8 |
| Prioridad | Necesario (**N**) |
| Validaciones | No aplica |
| Flujo alternativo | No aplica |
| Criterio de aceptación del usuario final | Los servicios de APIMRiesgo deben devolver la información en formato Json, donde necesariamente se debe detallar una descripción y un código HTTP. |

|  |  |
| --- | --- |
| **REQ # 06 - MICROSERVICIOS - API Management Empresa** | |
| Item: | 06 |
| Precondición | No aplica |
| Flujo normal | 1. Servicios: 2. Obtener todas las empresas   /api/listarEmpresas – GET  PARAMETERS:  Ninguno  RESPONSE:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Empresas encontradas correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  },  "empresa": [  {  "nombre": "DXC Technology",  "ruc": "20395886327",  "estado": 1  }  ]  }   1. Registrar una empresa:   /api/empresa – POST  REQUEST:  {  "nombre": "DXC Technology",  "ruc": "20395886327",  "estado": "1"  }  RESPONSE:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Empresa registrada correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  }  }   1. Editar una empresa:   /api/empresa - PUT  REQUEST:  {  "id": "4",  "nombre": "DXC Technology",  "ruc": "20395886327",  "estado": "1"  }  RESPONSE:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Empresa editada correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  }  }   1. Tecnologías de desarrollo:  * Spring Boot 2.2 * Spring Data JPA 2.2  1. Base de datos: MySQL 8 |
| Prioridad | Necesario (**N**) |
| Validaciones | No aplica |
| Flujo alternativo | No aplica |
| Criterio de aceptación del usuario final | Los servicios de APIMEmpresa deben devolver la información en formato Json, donde necesariamente se debe detallar una descripción y un código HTTP. |

|  |  |
| --- | --- |
| **REQ # 07 - MICROSERVICIOS - API Management Site** | |
| Item: | 07 |
| Precondición | No aplica |
| Flujo normal | 1. Servicios: 2. Obtener todos los sites   /api/listarSites – GET  PARAMETERS:  Ninguno  RESPONSE:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Sites encontrados correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  },  "site": [  {  "nombre": "Site 1",  }  ]  }   1. Registrar un site:   /api/site – POST  REQUEST:  {  "nombre": "Site 2"  }  RESPONSE:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Site registrado correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  }  }   1. Editar un site:   /api/site - PUT  REQUEST:  {  "id": "4"  "nombre": "Site 3"  }  RESPONSE:  {  "result": {  "code": 200,  "description": "Transacción correcta",  “descriptionDetail": "Site editado correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  }  }   1. Tecnologías de desarrollo:  * Spring Boot 2.2 * Spring Data JPA 2.2  1. Base de datos: MySQL 8 |
| Prioridad | Necesario (**N**) |
| Validaciones | No aplica |
| Flujo alternativo | No aplica |
| Criterio de aceptación del usuario final | Los servicios de APIMSite deben devolver la información en formato Json, donde necesariamente se debe detallar una descripción y un código HTTP. |

|  |  |
| --- | --- |
| **REQ # 08 - MICROSERVICIOS - API Management Nivel** | |
| Item: | 08 |
| Precondición | No aplica |
| Flujo normal | 1. Servicios: 2. Obtener todos los niveles   /api/listarNiveles – GET  PARAMETERS:  Ninguno  RESPONSE OK:  {  "result": {  "code": 200,  "description": "Transacción correcta",  "descriptionDetail": "Niveles encontrados correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  },  "nivel": [  {  "nombre": "Bajo",  }  ]  }   1. Registrar un nivel:   /api/nivel – POST  REQUEST:  {  "nombre": "Extremo"  }  RESPONSE OK:  {  "result": {  "code": 200,  "description": "Transacción correcta",  "descriptionDetail": "Nivel registrado correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  },  }   1. Editar un nivel:   /api/nivel - PUT  REQUEST:  {  "id": "4"  "nombre": "Extremo"  }  RESPONSE OK:  {  "result": {  "code": 200,  "description": "Transacción correcta",  "descriptionDetail": "Nivel editado correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  },  }   1. Tecnologías de desarrollo:  * Spring Boot 2.2 * Spring Data JPA 2.2  1. Base de datos: MySQL 8 |
| Prioridad | Necesario (**N**) |
| Validaciones | No aplica |
| Flujo alternativo | No aplica |
| Criterio de aceptación del usuario final | Los servicios de APIMNivel deben devolver la información en formato Json, donde necesariamente se debe detallar una descripción y un código HTTP. |

|  |  |
| --- | --- |
| **REQ # 09 - MICROSERVICIOS - API Management Rol** | |
| Item: | 09 |
| Precondición | No aplica |
| Flujo normal | 1. Servicios: 2. Obtener todos los roles   /api/listarRoles – GET  PARAMETERS:  Ninguno  RESPONSE OK:  {  "result": {  "code": 200,  "description": "Transacción correcta",  "descriptionDetail": "Roles encontrados correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  },  "rol": [  {  "nombre": "Sponsor",  }  ]  }   1. Registrar un rol:   /api/rol – POST  REQUEST:  {  "nombre": "Sponsor"  }  RESPONSE OK:  {  "result": {  "code": 200,  "description": "Transacción correcta",  "descriptionDetail": "Rol registrado correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  },  }   1. Editar un rol:   /api/rol - PUT  REQUEST:  {  "id": "2"  "nombre": "Operativo"  }  RESPONSE OK:  {  "result": {  "code": 200,  "description": "Transacción correcta",  "descriptionDetail": "Rol editado correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  },  }   1. Tecnologías de desarrollo:  * Spring Boot 2.2 * Spring Data JPA 2.2  1. Base de datos: MySQL 8 |
| Prioridad | Necesario (**N**) |
| Validaciones | No aplica |
| Flujo alternativo | No aplica |
| Criterio de aceptación del usuario final | Los servicios de APIMRol deben devolver la información en formato Json, donde necesariamente se debe detallar una descripción y un código HTTP. |

|  |  |
| --- | --- |
| **REQ # 10 - MICROSERVICIOS - API Management Token** | |
| Item: | 10 |
| Precondición | No aplica |
| Flujo normal | 1. Servicios: 2. Obtener un token generado que dure por un intervalo de tiempo:   /api/token/generate – POST  REQUEST:  {  "client-id":"1239234233044",  "secret-id": "23482340923423523"  }  RESPONSE OK:  {  "result": {  "code": 200,  "description": "Transacción correcta",  "descriptionDetail": "Token generado correctamente",  "timeStamp": "2020-04-26T16:56:59.000",  "transactionId": 125312647124  },  "token”: "23rmfermfro3495435tl34f-3454323-23294394"  }  COMENTARIOS DEL RESPONSE: Añadir el token en el parámetro “Authorization” del header de todos los microservicios.   1. Tecnologías de desarrollo:  * Spring Boot 2.2 * Spring Data JPA 2.2  1. Base de datos: MySQL 8 |
| Prioridad | Crítico (**C**) |
| Validaciones | No aplica |
| Flujo alternativo | No aplica |
| Criterio de aceptación del usuario final | Los servicios de APIMToken deben devolver la información en formato Json, donde necesariamente se debe detallar una descripción y un código HTTP. |