imexHS 2024

Manual técnico: Formularios dinámicos y cálculos de ANS - HubSpot

Contenido

Introducción

Este manual técnico proporciona una guía detallada sobre las herramientas y componentes utilizados para el desarrollo del formulario dinámico destinado a la radicación de tickets de soporte de empresas y distribuidores de IMEXHS, utilizando HubSpot.

Así como la configuración de horarios y tiempos de respuesta para los planes de soporte disponibles, además de detallar el funcionamiento de los cálculos y funciones para el conteo de periodos de resolución y validación del cumplimiento de estos según el ANS acordado con los clientes.

Descripción del proyecto

El proyecto está compuesto de 2 módulos que combinados trabajan en conjunto para permitir la gestión de los tickets de soporte de las empresas y distribuidores de IMEXHS además de llevar el cálculo los tiempos de respuesta de estos y el estado de cumplimiento de los ANS.

La primera parte se compone de un formulario dinámico que permite a los usuarios seleccionar el tipo de cliente que son (empresas o distribuidores) y escoger la empresa y sede desde la que se registra el ticket. Teniendo en cuenta que no todas las empresas o distribuidores están habilitados para realizar esta acción, por lo cual si no tienen autorización no se mostraran en los listados correspondientes.

La segunda parte se compone de una combinación de tablas en HubDB y código personalizado, dónde se realizan diferentes cálculos según el lapso de un ticket en estado abierto, y su duración en las diferentes etapas del pipeline, teniendo en cuenta los horarios de atención del nivel de soporte contratado por el cliente, para finalmente indicar si los tiempos acordados según éste se han cumplido o no.

Alcance de la documentación

Esta documentación abarca los aspectos necesarios para comprender el desarrollo e implementación del formulario dinámico destinado a la radicación de tickets de soporte de empresas y distribuidores de IMEXHS en HubSpot, así como la configuración de horarios y tiempos de respuesta para los planes de soporte disponibles.

1. **Configuración del Formulario Dinámico:**
    - Guía detallada para la personalización del formulario dinámico en HubSpot.
    - Procedimientos para la configuración de opciones condicionales basadas en el tipo de cliente (empresas o distribuidores) y la autorización para registrar tickets.
2. **Configuración de Horarios y Tiempos de Respuesta:**
    - Detalle sobre cómo definir y ajustar los horarios de atención en HubSpot usando HubDB
3. **Funcionamiento de Cálculos y Funciones:**
    - Explicación detallada del funcionamiento de los cálculos y funciones utilizadas para determinar los tiempos de resolución de los tickets.
    - Instrucciones paso a paso sobre la configuración de tablas en HubDB y código personalizado para realizar los cálculos.
4. **Validación del Cumplimiento de los ANS:**
    - Descripción de los criterios, propiedades y métricas utilizadas para validar el cumplimiento de los Acuerdos de Nivel de Servicio (ANS) acordados con los clientes.

Arquitectura y funcionamiento del sistema

Para el correcto funcionamiento de los módulos que componen el proceso de soporte a empresas y distribuidores, se utilizan los siguientes componentes de HubSpot:

1. Etiquetas:

Se utilizan las etiquetas de asociación entre empresas, de la siguiente manera:

- Child Company (Muchos): Indica las empresa subordinadas o filiales.
- Parent Company (Uno): Indica la empresa matriz o principal.

Estas etiquetas permiten que el sistema detecte cambios en la información de las empresas, Por ejemplo, cuándo se realiza un cambio en un objeto (como una Sede) asociada a la empresa, el sistema debería detectar dicho cambio y activar el Workflow para actualizar la información en HubDB. Sin embargo, el sistema no detecta estos cambios lo que impide que el Workflow se desencadene.

Con el uso adecuado de estas etiquetas, el sistema puede identificar cuándo una empresa distribuidora, está asociada a otra, lo que activa el Workflow necesario para lograr la actualización de información en HubDB.

2. HubDB:

Para el correcto funcionamiento del desarrollo se han creado las siguientes tablas en HubDB.

Empresas y Sedes | Formulario Soporte: Utilizada para almacenar la información de las empresas que se mostrarán como opciones de selección en el formulario dinámico.

Su estructura se compone de:

Nombre	Nombre interno	Tipo de dato	Descripción
ID	hs_id	Número	Identificador único autogenerado que se asigna automáticamente a cada registro.

ID interno (Empresa)	id_company	Número	Almacena el identificador de la empresa en el CRM de HubSpot.
Nombre	name	Texto	Almacena el nombre de la empresa.
JSON	json	Texto	Almacena en código JSON la estructura de las asociaciones del ítem, es decir, si cuenta con sedes asociadas, si es distribuidor y cuenta con empresas y sedes asociadas.

Datos ANS | Tickets: Contiene la configuración de los tipos de soporte prestados y los tiempos máximos de respuesta definidos según la prioridad del ticket. Se utiliza para definir las expectativas de servicio de acuerdo con las prioridades del soporte técnico.

Su estructura se compone de:

Nombre	Nombre interno	Tipo de dato	Descripción
ID	hs_id	Número	Identificador único autogenerado que se asigna automáticamente a cada registro.
Soporte dirigido a	soporte_dirigido_a	Texto	Define a qué tipo de contacto o cliente va dirigida la configuración del soporte
Tipo de soporte	tipo_de_soporte	Texto	Especifica los tipos de soporte que se prestan.
Tiempo en horas primera respuesta (Prioridad alta)	th_primera_rta_alta	Número	Número máximo de horas en las que se espera que el equipo de soporte dé la primera respuesta para tickets con prioridad alta
Tiempo en horas primera respuesta (Prioridad media)	th_primera_rta_media	Número	Número máximo de horas en las que se espera la primera respuesta para tickets de prioridad media.
Tiempo en horas primera respuesta (Prioridad baja)	th_primera_rta_baja	Número	Número máximo de horas en las que se debe proporcionar la primera respuesta para tickets de prioridad baja.
Tiempo en horas para cierre (Prioridad alta)	th_cierre_alta	Número	Tiempo máximo en horas para cerrar un ticket de prioridad alta. Este tiempo se refiere a la resolución completa del ticket.
Tiempo en horas para cierre (Prioridad media)	th_cierre_medio	Número	Tiempo máximo en horas para cerrar tickets con prioridad media.

Tiempo en horas para cierre (Prioridad baja)	th_cierre_baja	Número	Tiempo máximo en horas para cerrar tickets de baja prioridad.

Horarios atención: Contiene la configuración de los horarios de disponibilidad del personal de soporte según el tipo de servicio y los días de la semana. Estos horarios ayudan a establecer las expectativas de respuesta y soporte durante los días hábiles y fines de semana.

Su estructura se compone de:

Nombre	Nombre interno	Tipo de dato	Descripción
ID	hs_id	Número	Identificador único autogenerado que se asigna automáticamente a cada registro.
Lunes	lunes	Hora hh:mm (Hora entrada;Hora salida)	Rango de horas disponibles para brindar soporte el lunes
Martes	martes	Hora hh:mm (Hora entrada;Hora salida)	Rango de horas disponibles para brindar soporte el martes
Miércoles	miercoles	Hora hh:mm (Hora entrada;Hora salida)	Rango de horas disponibles para brindar soporte el miércoles
Jueves	jueves	Hora hh:mm (Hora entrada;Hora salida)	Rango de horas disponibles para brindar soporte el jueves
Viernes	viernes	Hora hh:mm (Hora entrada;Hora salida)	Rango de horas disponibles para brindar soporte el viernes
Sábado	sabado	Hora hh:mm (Hora entrada;Hora salida)	Rango de horas disponibles para brindar soporte el sábado
Domingo	domingo	Hora hh:mm (Hora entrada;Hora salida)	Rango de horas disponibles para brindar soporte el domingo
Relación ANS	relacion_ans	Número (Clave foránea)	Relación con la tabla “**Datos ANS | Tickets”**. Vincula los horarios con las configuraciones de los tipos de soporte.

IMPORTANTE: Aunque las tablas están diseñadas para ser flexibles y permitir actualizaciones rápidas en las configuraciones de ANS y horarios según las necesidades de los clientes, es crucial asegurarse de que las relaciones entre las

tablas estén correctamente definidas. En particular, se debe prestar atención a la columna **RELACION_ANS** de la tabla **Horarios Atención** para garantizar que cada horario de atención está asociado al tipo de soporte y prioridad correctos.

3. Formularios:

El funcionamiento para la radicación de tickets está compuesto de siete (7) formularios, uno (1) que se puede considerar el principal: **“Formulario Desencadenante – Soporte”** y seis (6) secundarios que se usan para limitar el comportamiento del formulario principal según el plan de soporte contratado por la empresa o distribuidor. Estos se pueden encontrar en la carpeta **“Formulario Dinámico Tickets”**

El formulario **“Desencadenante – Soporte”**, se usa como plantilla principal, para generar los campos dinámicos. Es decir, se usa el código de incrustación para conservar la funcionalidad del sistema de HubSpot y se generan los campos dinámicamente. Por lo que el formulario no se usa para realizar envíos, si no, que se utiliza para “imitar” el comportamiento de un formulario nativo de HubSpot.

El campo de distribuidor, empresa y sede, se alimentan de la tabla **Empresas y Sedes | Formulario Soporte** de HubDB, por medio de diferentes funciones según sea la elección del usuario y el nivel de soporte, el sistema renderiza uno (1) de los formularios de "**Clientes directos y Distribuidores**". Permitiendo así, que el ticket que se diligencie ingrese en la bandeja correspondiente a su nivel de soporte.

IMPORTANTE: Las siguientes propiedades de los formularios de **Clientes directos** y **Distribuidores** no pueden alterarse o eliminarse, ya que podría afectar el funcionamiento del sistema de forma critica.

4. Workflows:

El funcionamiento para la radicación de tickets está compuesto de nueve (9) Workflows. Estos se pueden encontrar en la carpeta **“Formulario Dinámico Tickets”**

Objeto empresas:

Actualización información Empresa a Empresa HubDB | Empresas (Funciones nativas): Actualiza la propiedad "Última modificación sede" cuando se conoce la propiedad, y la empresa está autorizada para el formulario, además de no ser un distribuidor.

Actualización HubDB | Empresas (Código personalizado): Actualiza la tabla de HubDB con el ID de registro de la empresa inscrita cuando la propiedad "Autorizada para formulario" sea "Sí" y exista una fecha en la propiedad "Fecha de última modificación".

```
const { IMEXHS_API_KEY } = process.env
const axios = require('axios');
const URL_API    = "https://api.hubapi.com"
const HEADERS = {
    'authorization': `Bearer ${IMEXHS_API_KEY}`,
    'Content-Type': 'application/json'
}
const sedesObject = "2-24332058"


exports.main = async (event, callback) => {

  // Varibles
  const company_id = event.inputFields['company_id']
  let item_array = []
  let sedes_list

  // Revisa si la Empresa existe en HubDB
  let resGetRowTable = await getRowTable(company_id)
  if(!resGetRowTable){
    console.log("Empresa no existente en HubDB")
    return
  }

  // Obtiene toda la información de la Empresa
  let resCompany = await getCompany(company_id)
  if(!resCompany){
    console.log("La Empresa no existe")
    return
  }

```

```
  // Organiza la información necesaria de la empresa en un nuevo Objeto
  item_array = {
    id: resCompany.properties.hs_object_id,
    name: resCompany.properties.name,
    tipo_de_soporte: resCompany.properties.tipo_de_soporte,
    es_distribuidor: resCompany.properties.es_distribuidor
   }


  // Verifica SI la Empresa esta marcada como "Distribuidor" & Si tiene Asociaciones & Si esa asociacion tiene empresas
  if(item_array.es_distribuidor == "true" && typeof resCompany.associations !== "undefined" && typeof resCompany.associations.companies !== "undefined") {
    // Inicializa el Objeto Empresas
    item_array.empresas = []

    // Recorre la informacion de las compañias asociadas
    for(const item of resCompany.associations.companies.results){
      let tempCompany = {}

      // Valida si presenta la etiqueta de asociacion de empresa principal o secundaria
      if(item.type === "parent_to_child_company"){

        // Consulta la informacion completa de cada empresa
        let dataCompany = await getCompany(item.id)
        if(!dataCompany){
          console.log("La empresa no existe")
          return
        }

        // Organiza los datos de la empresa en un nuevo objeto
        tempCompany = {
          id: dataCompany.properties.hs_object_id,
          name: dataCompany.properties.name,
          tipo_de_soporte: dataCompany.properties.tipo_de_soporte,
        }

        // Verifica si cuenta con Sedes asociadas
        if(typeof dataCompany.associations.p8943891_sedes != "undefined"){
          // Extrae el ID de las sedes
          let companySedesList = dataCompany.associations.p8943891_sedes.results.map(sede => ({
            id: sede.id
          }))

          // Consulta la informacion de las Sedes en lote
          let resCompanySede = await getSedes(companySedesList)
          if(!resCompanySede){
            console.log("las SEDES no existen")
            return
          }
```

```
81.
82.           // Inicializa el Objeto de Sedes
83.           tempCompany.sedes = []
84.
85.           // Carga la informacion de las sedes sobre el Objeto Sedes
86.           resCompanySede.forEach(data => {
87.             tempCompany.sedes.push({
88.               id: data.id,
89.               name: data.name,
90.               tipo_de_soporte: data.tipo_de_soporte
91.             });
92.           });
93.
94.           // Carga la informacion sobre el Objeto ordenado principal
95.           item_array.empresas.push(tempCompany)
96.         }else{
97.
98.           // Si no encuentra Sedes carga solamente la informacion de la
   empresa
99.           item_array.empresas.push(tempCompany)
100.              }
101.            }
102.          }
103.        }
104.
105.
106.        // Verifica SI la Empresa NO esta marcada como "Distribuidor" &
   Si el campo de distribuidor esta vacio
107.        if(item_array.es_distribuidor == undefined ||
   item_array.es_distribuidor == "" || item_array.es_distribuidor ==
   "false"){
108.
109.          //Verifica que no tenga asociaciones con distribuidores
110.          if(typeof resCompany.associations !== "undefined" && typeof
   resCompany.associations.companies !== "undefined"){
111.
112.            // Elimina el registro de Empresa en HubDB
113.            let resCreateRow = await deleteRowTable(resGetRowTable.id)
114.            if(!resCreateRow){
115.              console.log("No se pudo eliminar fila en tabla")
116.              return
117.            }
118.
119.            // Publica la tabla de HubDB
120.            let resPublishTable = await publishTable()
121.            if(!resPublishTable){
122.              console.log("No se publico la tabla")
123.              return
124.            }
125.
126.            console.log("las EMPRESA ya pertenece a un distribuidor")
127.            return
128.          }
129.
130.          // Verifica si la Empresa tiene asociaciones de sedes
```

```
131.         if(typeof resCompany.associations !== "undefined" && typeof
   resCompany.associations.p8943891_sedes !== "undefined"){
132.           // Extrae el ID de las sedes
133.           sedes_list =
   resCompany.associations.p8943891_sedes.results.map(item => ({
134.             id: item.id
135.           }))
136.
137.           // Consulta la informacion de las Sedes en lote
138.           let resSede = await getSedes(sedes_list)
139.           if(!resSede){
140.             console.log("las SEDES no existen")
141.             return
142.           }
143.
144.           // Inicializa el Objeto de Sedes sobre el objeto principal
145.           item_array.sedes = [];
146.
147.           // Carga la informacion de las sedes sobre el Objeto Sedes
148.           resSede.forEach(data => {
149.             item_array.sedes.push({
150.               id: data.id,
151.               name: data.name,
152.               tipo_de_soporte: data.tipo_de_soporte
153.             });
154.           });
155.         } else {
156.           console.log("La empresa no tiene ninguna ASOCIACIÓN")
157.         }
158.       }
159.
160.       // Actualiza el registro de la empresa en la tabla de HubDB
161.       let resUpdateRowTable = await
   updateRowTable(item_array,resGetRowTable.id)
162.       if(!resUpdateRowTable){
163.         console.log("No actualizo registro en tabla ")
164.         return
165.       }
166.
167.       // Publica la tabla de HubDB
168.       let resPublishTable = await publishTable()
169.       if(!resPublishTable){
170.         console.log("No se publico la tabla")
171.         return
172.       }
173.
174.       /*callback({
175.         outputFields: {
176.           email: email
177.         }
178.       });*/
179.     }
180.
181.     /**
182.      * Consultar Empresa
```

```
183.       * @param {*} companyId
184.       */
185.      async function getCompany(companyId){
186.        const config = {
187.          method: 'GET',
188.          url: URL_API +
    `/crm/v3/objects/companies/${companyId}?associations=${sedesObject}&assoc
    iations=company&properties=id,name,tipo_de_soporte,es_distribuidor`,
189.          headers: HEADERS
190.        };
191.        try {
192.          let req = await axios(config)
193.          let res = req.data
194.          if(res){
195.            return res
196.          }
197.          else{
198.            return false
199.          }
200.        } catch (error) {
201.          return false
202.        }
203.      }
204.
205.
206.      /**
207.       * Consultar fila HubDB
208.       * @param {*} companyId
209.       */
210.      async function getRowTable(companyId){
211.        const config = {
212.          method: 'GET',
213.          url: URL_API +
    `/cms/v3/hubdb/tables/companies_imexhs/rows?id_company__eq=${companyId}`,
214.          headers: HEADERS
215.        };
216.        try {
217.          let req = await axios(config)
218.          let res = req.data
219.          if(res.total != 0){
220.            return res.results[0]
221.          }
222.          else{
223.            return false
224.          }
225.        } catch (error) {
226.          return false
227.        }
228.      }
229.
230.      /**
231.       * Consultar Sedes
232.       * @param {*} sedeList
233.       */
234.      async function getSedes(sedeList){
```

```
235.         var data = JSON.stringify({
236.           "propertiesWithHistory": [
237.             "nombre_de_sede",
238.             "tipo_de_soporte"
239.           ],
240.           "inputs": sedeList,
241.           "properties": [
242.             "nombre_de_sede",
243.             "tipo_de_soporte"
244.           ]
245.         });
246.
247.         const config = {
248.           method: 'POST',
249.           url: URL_API + `/crm/v3/objects/${sedesObject}/batch/read`,
250.           headers: HEADERS,
251.           data: data
252.         };
253.         try {
254.           let req = await axios(config)
255.           let res = req.data
256.           if(res){
257.             return res.results.map(item => ({
258.               id: item.properties.hs_object_id,
259.               name: item.properties.nombre_de_sede,
260.               tipo_de_soporte: item.properties.tipo_de_soporte
261.             }))
262.           }
263.           else{
264.             return false
265.           }
266.         } catch (error) {
267.           return false
268.         }
269.       }
270.       /**
271.        * Crear fila HubDB
272.        * @param {*} code
273.        * @param {*} rowId
274.        */
275.       async function updateRowTable(code,rowId){
276.         var data = JSON.stringify({
277.           "values": {
278.             "id_company": parseInt(code.id),
279.             "name": code.name,
280.             "json": JSON.stringify(code)
281.           },
282.         });
283.
284.         const config = {
285.           method: 'PATCH',
286.           url: URL_API +
    `/cms/v3/hubdb/tables/companies_imexhs/rows/${rowId}/draft`,
287.           headers: HEADERS,
288.           data: data
```

```
289.         };
290.         try {
291.           let req = await axios(config)
292.           let res = req.data
293.           if(res){
294.             return res
295.           }
296.           else{
297.             return false
298.           }
299.         } catch (error) {
300.           return false
301.         }
302.       }
303.
304.       /**
305.        * Borrar fila HubDB
306.        * @param {*} companyId
307.        */
308.       async function deleteRowTable(rowId){
309.         const config = {
310.           method: 'DELETE',
311.           url: URL_API +
    `/cms/v3/hubdb/tables/companies_imexhs/rows/${rowId}/draft`,
312.           headers: HEADERS
313.         };
314.         try {
315.           let req = await axios(config)
316.           let res = req.data
317.           return true
318.         } catch (error) {
319.           return false
320.         }
321.       }
322.       /**
323.        * Publicar tabla HubDB
324.        *
325.        */
326.       async function publishTable(){
327.
328.         const config = {
329.           method: 'POST',
330.           url: URL_API +
    '/cms/v3/hubdb/tables/companies_imexhs/draft/publish',
331.           headers: HEADERS
332.         };
333.         try {
334.           let req = await axios(config)
335.           let res = req.data
336.           if(res){
337.             return res
338.           }
339.           else{
340.             return false
341.           }
```

```
        } catch (error) {
          return false
        }
      }
```

Gestión Empresa y Sedes | Tickets | HubDB (Código personalizado): Agrega o elimina la empresa o sede de HubDB según la configuración de la propiedad “Autorizado para formulario”

Agregar empresa en HubDB:

```
const { IMEXHS_API_KEY } = process.env
const axios = require('axios');
const URL_API    = "https://api.hubapi.com"
const HEADERS = {
    'authorization': `Bearer ${IMEXHS_API_KEY}`,
    'Content-Type': 'application/json'
}
const sedesObject = "2-24332058"


exports.main = async (event, callback) => {

  // Varibles
  const company_id = event.inputFields['company_id']
  let item_array = []
  let sedes_list

  // Revisa si la Empresa existe en HubDB
  let resGetRowTable = await getRowTable(company_id)
  if(resGetRowTable){
    console.log("Empresa existente en HubDB")
    return
  }

  // Obtiene toda la información de la Empresa
  let resCompany = await getCompany(company_id)
  if(!resCompany){
    console.log("La empresa no existe")
    return
  }

  // Organiza la información necesaria de la empresa en un nuevo Objeto
  item_array = {
    id: resCompany.properties.hs_object_id,
    name: resCompany.properties.name,
    tipo_de_soporte: resCompany.properties.tipo_de_soporte,
    es_distribuidor: resCompany.properties.es_distribuidor
   }

  // Verifica SI la Empresa esta marcada como "Distribuidor" & Si tiene Asociaciones & Si esa asociacion tiene empresas
```

```
  if(item_array.es_distribuidor == "true" && typeof resCompany.associations !== "undefined" && typeof resCompany.associations.companies !== "undefined") {
    // Inicializa el Objeto Empresas
    item_array.empresas = []

    // Recorre la informacion de las compañias asociadas
    for(const item of resCompany.associations.companies.results){
      let tempCompany = {}

      // Valida si presenta la etiqueta de asociacion de empresa principal o secundaria
      if(item.type === "parent_to_child_company"){

        // Consulta la informacion completa de cada empresa
        let dataCompany = await getCompany(item.id)
        if(!dataCompany){
          console.log("La empresa no existe")
          return
        }

        // Organiza los datos de la empresa en un nuevo objeto
        tempCompany = {
          id: dataCompany.properties.hs_object_id,
          name: dataCompany.properties.name,
          tipo_de_soporte: dataCompany.properties.tipo_de_soporte,
        }

        // Verifica si cuenta con Sedes asociadas
        if(typeof dataCompany.associations.p8943891_sedes != "undefined"){
          // Extrae el ID de las sedes
          let companySedesList = dataCompany.associations.p8943891_sedes.results.map(sede => ({
            id: sede.id
          }))

          // Consulta la informacion de las Sedes en lote
          let resCompanySede = await getSedes(companySedesList)
          if(!resCompanySede){
            console.log("las SEDES no existen")
            return
          }

          // Inicializa el Objeto de Sedes
          tempCompany.sedes = []

          // Carga la informacion de las sedes sobre el Objeto Sedes
          resCompanySede.forEach(data => {
            tempCompany.sedes.push({
              id: data.id,
              name: data.name,
              tipo_de_soporte: data.tipo_de_soporte
            });
          });
```

```
91.
92.           // Carga la informacion sobre el Objeto ordenado principal
93.           item_array.empresas.push(tempCompany)
94.         }else{
95.
96.           // Si no encuentra Sedes carga solamente la informacion de la
   empresa
97.           item_array.empresas.push(tempCompany)
98.         }
99.       }
100.          }
101.        }
102.
103.        // Verifica SI la Empresa NO esta marcada como "Distribuidor" &
   Si el campo de distribuidor esta vacio
104.        if(item_array.es_distribuidor == undefined ||
   item_array.es_distribuidor == "" || item_array.es_distribuidor ==
   "false"){
105.
106.          //Verifica que no tenga asociaciones con distribuidores
107.          if(typeof resCompany.associations !== "undefined" && typeof
   resCompany.associations.companies !== "undefined"){
108.            console.log("las EMPRESA ya pertenece a un distribuidor")
109.            return
110.          }
111.
112.          // Verifica si la Empresa tiene asociaciones de sedes
113.          if(typeof resCompany.associations !== "undefined" && typeof
   resCompany.associations.p8943891_sedes !== "undefined"){
114.            // Extrae el ID de las sedes
115.            sedes_list =
   resCompany.associations.p8943891_sedes.results.map(item => ({
116.              id: item.id
117.            }))
118.
119.            // Consulta la informacion de las Sedes en lote
120.            let resSede = await getSedes(sedes_list)
121.            if(!resSede){
122.              console.log("las SEDES no existen")
123.              return
124.            }
125.
126.            // Inicializa el Objeto de Sedes sobre el objeto principal
127.            item_array.sedes = [];
128.
129.            // Carga la informacion de las sedes sobre el Objeto Sedes
130.            resSede.forEach(data => {
131.              item_array.sedes.push({
132.                id: data.id,
133.                name: data.name,
134.                tipo_de_soporte: data.tipo_de_soporte
135.              });
136.            });
137.          } else {
138.            console.log("La empresa no tiene ninguna ASOCIACIÓN")
```

```
        }
      }

      // Actualiza el registro de la empresa en la tabla de HubDB
      let resCreateRow = await createRowTable(item_array)
      if(!resCreateRow){
        console.log("No se creo fila en tabla ")
        return
      }

      // Publica la tabla de HubDB
      let resPublishTable = await publishTable()
      if(!resPublishTable){
        console.log("No se publico la tabla")
        return
      }

      /*callback({
        outputFields: {
          email: email
        }
      });*/
    }

    /**
     * Consultar Empresa
     * @param {*} companyId
     */
    async function getCompany(companyId){
      const config = {
        method: 'GET',
        url: URL_API +
  `/crm/v3/objects/companies/${companyId}?associations=${sedesObject}&assoc
  iations=company&properties=id,name,tipo_de_soporte,es_distribuidor`,
        headers: HEADERS
      };
      try {
        let req = await axios(config)
        let res = req.data
        if(res){
          return res
        }
        else{
          return false
        }
      } catch (error) {
        return false
      }
    }

    /**
     * Consultar Sedes
     * @param {*} sedeList
     */
    async function getSedes(sedeList){
```

```
192.        var data = JSON.stringify({
193.          "propertiesWithHistory": [
194.            "nombre_de_sede",
195.            "tipo_de_soporte"
196.          ],
197.          "inputs": sedeList,
198.          "properties": [
199.            "nombre_de_sede",
200.            "tipo_de_soporte"
201.          ]
202.        });
203.
204.        const config = {
205.          method: 'POST',
206.          url: URL_API + `/crm/v3/objects/${sedesObject}/batch/read`,
207.          headers: HEADERS,
208.          data: data
209.        };
210.        try {
211.          let req = await axios(config)
212.          let res = req.data
213.          if(res){
214.            return res.results.map(item => ({
215.              id: item.properties.hs_object_id,
216.              name: item.properties.nombre_de_sede,
217.              tipo_de_soporte: item.properties.tipo_de_soporte
218.            }))
219.          }
220.          else{
221.            return false
222.          }
223.        } catch (error) {
224.          return false
225.        }
226.      }
227.
228.      /**
229.       * Consultar fila HubDB
230.       * @param {*} companyId
231.       */
232.      async function getRowTable(companyId){
233.        const config = {
234.          method: 'GET',
235.          url: URL_API +
    `/cms/v3/hubdb/tables/companies_imexhs/rows?id_company__eq=${companyId}`,
236.          headers: HEADERS
237.        };
238.        try {
239.          let req = await axios(config)
240.          let res = req.data
241.          if(res.total != 0){
242.            return true
243.          }
244.          else{
245.            return false
```

```
246.         }
247.       } catch (error) {
248.         return false
249.       }
250.     }
251.     /**
252.      * Crear fila HubDB
253.      * @param {*} code
254.      */
255.     async function createRowTable(code){
256.       var data = JSON.stringify({
257.         "values": {
258.           "id_company": parseInt(code.id),
259.           "name": code.name,
260.           "json": JSON.stringify(code)
261.         },
262.       });
263.
264.       const config = {
265.         method: 'POST',
266.         url: URL_API + '/cms/v3/hubdb/tables/companies_imexhs/rows',
267.         headers: HEADERS,
268.         data: data
269.       };
270.       try {
271.         let req = await axios(config)
272.         let res = req.data
273.         if(res){
274.           return res
275.         }
276.         else{
277.           return false
278.         }
279.       } catch (error) {
280.         return false
281.       }
282.     }
283.
284.     /**
285.      * Publicar tabla HubDB
286.      */
287.     async function publishTable(){
288.
289.       const config = {
290.         method: 'POST',
291.         url: URL_API +
    '/cms/v3/hubdb/tables/companies_imexhs/draft/publish',
292.         headers: HEADERS
293.       };
294.       try {
295.         let req = await axios(config)
296.         let res = req.data
297.         if(res){
298.           return res
299.         }
```

```
          else{
            return false
          }
        } catch (error) {
          return false
        }
      }
```

Eliminar empresa de HubDB:

```
const { IMEXHS_API_KEY } = process.env
const axios = require('axios');
const URL_API   = "https://api.hubapi.com"
const HEADERS = {
    'authorization': `Bearer ${IMEXHS_API_KEY}`,
    'Content-Type': 'application/json'
}

exports.main = async (event, callback) => {

        const company_id = event.inputFields['company_id']

        // Revisa si la Empresa existe en HubDB
        let resGetRowTable = await getRowTable(company_id)
        if(!resGetRowTable){
          console.log("Empresa no existente en HubDB")
          return
        }

        // Elimina el registro de Empresa en HubDB
        let resCreateRow = await deleteRowTable(resGetRowTable.id)
        if(!resCreateRow){
          console.log("No se pudo eliminar fila en tabla")
          return
        }

        // Publica la tabla de HubDB
        let resPublishTable = await publishTable()
        if(!resPublishTable){
          console.log("No se publico la tabla")
          return
        }

        /*callback({
          outputFields: {
            email: email
```

```
        }
      });*/
    }

    /**
     * Consultar fila HubDB
     * @param {*} companyId
     */
    async function getRowTable(companyId){
      const config = {
        method: 'GET',
        url: URL_API + `/cms/v3/hubdb/tables/companies_imexhs/rows?id_company__eq=${companyId}`,
        headers: HEADERS
      };
      try {
        let req = await axios(config)
        let res = req.data
        if(res.total != 0){
          return res.results[0]
        }
        else{
          return false
        }
      } catch (error) {
        return false
      }
    }

    /**
     * Borrar fila HubDB
     * @param {*} companyId
     */
    async function deleteRowTable(rowId){
      const config = {
        method: 'DELETE',
        url: URL_API + `/cms/v3/hubdb/tables/companies_imexhs/rows/${rowId}/draft`,
        headers: HEADERS
      };
      try {
        let req = await axios(config)
        let res = req.data
        return true
      } catch (error) {
        return false
      }
    }
```

```
83.
84.       /**
85.        * Publicar tabla HubDB
86.        * @param {*} companyId
87.        */
88.       async function publishTable(){
89.
90.         const config = {
91.           method: 'POST',
92.           url: URL_API +
   '/cms/v3/hubdb/tables/companies_imexhs/draft/publish',
93.           headers: HEADERS
94.         };
95.         try {
96.           let req = await axios(config)
97.           let res = req.data
98.           if(res){
99.             return res
100.          }
101.          else{
102.            return false
103.          }
104.        } catch (error) {
105.          return false
106.        }
107.     }
```

Objeto sedes:

Actualización HubDB | Sedes (Código personalizado): Actualiza la propiedad "Última modificación sede" con la marca de tiempo actual cuando se conoce la fecha y hora de la última modificación del objeto.

```
1. from datetime import datetime
2. def main(event):
3.   now = datetime.now()
4.   unix_timestamp = int(now.timestamp())
5.   return {
6.     "outputFields": {
7.       "fecha": unix_timestamp
8.     }
9.   }
```

Objeto tickets:

Asociación Ticket a Empresa (Código personalizado): Asocia un ticket a una empresa cuando se conoce la propiedad "Empresa", después de un retraso de 1 minuto.

```
const { IMEXHS_API_KEY } = process.env
const axios = require('axios');
const URL_API   = "https://api.hubapi.com"
const HEADERS = {
    'authorization': `Bearer ${IMEXHS_API_KEY}`,
    'Content-Type': 'application/json'
}

exports.main = async (event,callback) => {

        const ticketId = event.inputFields['ticket_id'];
        const companyId = Number(event.inputFields['company_id'].split("|")[0]);

        if(!isNaN(companyId)){
          let assocCompany = await associateTicketToCompany(ticketId,companyId)
          if(!assocCompany){
            console.log("La empresa no existe")
            return
          }

         callback({
          outputFields: {
            successful: true
          }
         });

        }else{
          console.log("No se encontro ID de Empresa")
          return
        }
    }

    /**
     * Asociar Empresa a Ticket
     * @param {object}
     */
    async function associateTicketToCompany(ticketId,companyId){

        const data = JSON.stringify([
```

```
                {
                  "associationCategory": "HUBSPOT_DEFINED",
                  "associationTypeId": 26
                }
              ]);

          const config = {
              method: 'PUT',
              url: URL_API+`/crm/v4/objects/ticket/${ticketId}/associations/company/${companyId}`,
              headers: HEADERS,
              data : data
          };

          try {
              let req = await axios(config)
              let res = req.data
              if(res.status == "error"){
                  return false
              }

              return true

          } catch (error) {
              return false
          }
      }
```

Asociación Ticket a Sede (Código personalizado): Actualiza un ticket asociándolo a una empresa cuando se conoce la propiedad "Sede".

```
const { IMEXHS_API_KEY } = process.env
const axios = require('axios');
const URL_API   = "https://api.hubapi.com"
const HEADERS = {
    'authorization': `Bearer ${IMEXHS_API_KEY}`,
    'Content-Type': 'application/json'
}
const sedesObject = "2-24332058"

      exports.main = async (event,callback) => {

        const ticketId = event.inputFields['ticket_id'];
        const sedeId = Number(event.inputFields['sede_id'].split("|")[0]);
```

```
14.
15.         if(!isNaN(sedeId)){
16.           let assocCompany = await
    associateTicketToCompany(ticketId,sedeId,sedesObject)
17.           if(!assocCompany){
18.             console.log("La Sede no existe")
19.             return
20.           }
21.
22.          callback({
23.           outputFields: {
24.             successful: true
25.           }
26.          });
27.
28.         }else{
29.           console.log("No se encontro ID de Sede")
30.           return
31.         }
32.       }
33.
34.       /**
35.        * Asociar Sede a Ticket
36.        * @param {object}
37.        */
38.       async function associateTicketToCompany(ticketId,sedeId,object){
39.
40.           const data = JSON.stringify([
41.                 {
42.                   "associationCategory": "USER_DEFINED",
43.                   "associationTypeId": 230
44.                 }
45.               ]);
46.
47.           const config = {
48.               method: 'PUT',
49.               url:
    URL_API+`/crm/v4/objects/ticket/${ticketId}/associations/${object}/${sede
    Id}`,
50.               headers: HEADERS,
51.               data : data
52.           };
53.
54.           try {
55.               let req = await axios(config)
56.               let res = req.data
57.               if(res.status == "error"){
58.                   return false
```

```
            }

            return true

        } catch (error) {
            return false
        }
    }
```

Contar horas trabajadas en ticket abierto (Código personalizado): Cuenta las horas que se trabajaron en el ticket, debe estar en abierto para hacer el cálculo.

Función *main* (event)

Esta función es la entrada principal del programa y se espera que sea invocada con un evento como argumento. El evento contiene información relevante sobre el ticket de servicio.

Parámetros:

- `event`: Un diccionario que contiene información sobre el ticket de servicio.

Variables utilizadas:

- `idTicket`: Identificador único del ticket de servicio.
- `tipoSoporte`: Tipo de soporte asociado al ticket.
- `pipeline_stage`: Etapa actual del pipeline del ticket.
- `pipeline`: Pipeline al que pertenece el ticket.
- `hs_ticket_priority`: Prioridad del ticket.
- `es_distribuidor`: Indica si el ticket está dirigido a un distribuidor.
- `area_de_escalamiento`: Área de escalamiento del ticket.
- `dirigidoA`: Indica si el ticket está dirigido al cliente o a un distribuidor.
- `client`: Cliente de la API de HubSpot.
- `ApiResponse`: Respuesta de la API de HubSpot al obtener información del ticket.
- `stage`: Etapa del pipeline del ticket.
- `desplazamiento_colombia`: Diferencia de tiempo para ajustar a la zona horaria de Colombia.
- `fechainicial`: Fecha inicial del ticket.
- `stageInfo`: Información de la etapa del pipeline del ticket.
- `ans`: Información del soporte obtenida de HubDB.

- `ansInfo`: Información específica del soporte.
- `ansHorario`: Horario laboral obtenido de HubDB.
- `ansHorarioInfo`: Información del horario laboral.
- `nombres_dias`: Nombres de los días de la semana.
- `hoy`: Fecha y hora actual.
- `sumaDehoras`: Suma total de horas laboradas.
- `auxFechaInicio`: Fecha inicial auxiliar.
- `totalhoras`: Total de horas trabajadas.
- `namePropertyTime`: Nombre de la propiedad de tiempo del ticket.
- `properties`: Diccionario de propiedades del ticket.
- `tAreaEscalado`: Tiempo de escalado del área.
- `updateTicket`: Actualización del ticket.

Flujo de ejecución:

1. Se recuperan los parámetros del ticket del evento.
2. Se crea un cliente para la API de HubSpot.
3. Se obtiene la información del ticket de la API de HubSpot.
4. Se calcula la fecha inicial del ticket.
5. Se obtiene información del tipo de soporte y del horario laboral.
6. Se calcula la suma total de horas laboradas hasta la fecha actual.
7. Se calcula la suma de horas laboradas para el día actual.
8. Se actualiza el ticket con el tiempo total trabajado.

Consideraciones:

- Este código hace uso de la API de HubSpot para obtener y actualizar información de tickets de servicio.
- Se realizan cálculos de tiempo para determinar las horas laboradas.
- Se asume una diferencia de tiempo para ajustarse a la zona horaria de Colombia.
- Se utilizan datos almacenados en HubDB para obtener información sobre el tipo de soporte y el horario laboral.

```
1. import os
2. import hubspot
3. from hubspot.crm.tickets import ApiException,SimplePublicObjectInput
4. from datetime import datetime,timedelta,timezone
5.
6. #------------------------------------------------------------------------
   ---------#
7. def main(event):
8.
```

```
    num_stage = 0 #Numero de etapa a calcular en orden descendente (0 Ultima Etapa)
    idTicket = event["inputFields"]["hs_ticket_id"]
    tipoSoporte = event["inputFields"]["tipo_soporte"]
    pipeline_stage = event["inputFields"]["hs_pipeline_stage"]
    pipeline = event["inputFields"]["hs_pipeline"]
    hs_ticket_priority = event["inputFields"]["hs_ticket_priority"]
    es_distribuidor = "false"
    if "es_distribuidor" in event["inputFields"]:
        es_distribuidor = event["inputFields"]["es_distribuidor"]

    area_de_escalamiento = ""
    if "area_de_escalamiento" in event["inputFields"]:
        area_de_escalamiento = event["inputFields"]["area_de_escalamiento"]

    dirigidoA = ""
    if es_distribuidor == "true":
        dirigidoA = "Distribuidor"
    else:
        dirigidoA = "Cliente"

    client = hubspot.Client.create(access_token=os.getenv('ANS_API_KEY'))
    ApiResponse = client.crm.tickets.basic_api.get_by_id(ticket_id=idTicket, properties=["hs_pipeline_stage"], properties_with_history=["hs_pipeline_stage"], archived=False)
    stage = ApiResponse.properties_with_history.get('hs_pipeline_stage')

    if len(stage) > 1:

        desplazamiento_colombia = timedelta(hours=-5)
        hoy = stage[num_stage].timestamp+desplazamiento_colombia
        fechainicial = stage[num_stage+1].timestamp+desplazamiento_colombia
        # fechainicialDate=datetime.strptime(fechainicial, "%Y-%m-%dT%H:%M:%S.%fZ")
        stageInfo = client.crm.pipelines.pipeline_stages_api.get_by_id(
            object_type="tickets", pipeline_id=pipeline, stage_id=stage[num_stage+1].value)

        ans = client.cms.hubdb.rows_api.get_table_rows(
            table_id_or_name="15666306")
        ansInfo = {}
        for an in ans.results:
            if an.values["tipo_de_soporte"].lower() == tipoSoporte.lower() and dirigidoA.lower() == an.values["soporte_dirigido_a"].lower():
                ansInfo = {"id": an.id,
                           "tipo_de_soporte": an.values["tipo_de_soporte"],
                           "soporte_dirigido_a": an.values["soporte_dirigido_a"],
                           "th_primera_rta_baja": an.values["th_primera_rta_baja"],
```

```
                        "th_primera_rta_media":
an.values["th_primera_rta_media"],
                        "th_primera_rta_alta":
an.values["th_primera_rta_alta"],
                        "th_cierre_baja": an.values["th_cierre_baja"],
                        "th_cierre_medio": an.values["th_cierre_medio"],
                        "th_cierre_alta": an.values["th_cierre_alta"]}

        ansHorario = client.cms.hubdb.rows_api.get_table_rows(
            table_id_or_name="15666307")
        ansHorarioInfo = {}
        for ansH in ansHorario.results:
            # print(ansInfo)
            if ansH.values["relacion_ans"][0]["id"] == ansInfo["id"]:
                ansHorarioInfo = ansH.values

        nombres_dias = ["lunes", "martes", "miercoles",
                        "jueves", "viernes", "sabado", "domingo"]
        sumaDehoras = 0
        auxFechaInicio =
(stage[num_stage+1].timestamp+desplazamiento_colombia)

        #Calcular horas laboradas totales entre Pipeline
        while fechainicial.date() < hoy.date():
            if nombres_dias[fechainicial.weekday()] in ansHorarioInfo:
                horarioDia =
ansHorarioInfo[nombres_dias[fechainicial.weekday()]].split(
                    ";")
                if auxFechaInicio.date() == fechainicial.date():
                    horainicio = fechainicial.strftime('%H:%M')
                    horainicio = max(datetime.strptime(
                        horarioDia[0], '%H:%M'),
datetime.strptime(horainicio, '%H:%M'))
                else:
                    horainicio = datetime.strptime(horarioDia[0],
'%H:%M')
                horafin = datetime.strptime(horarioDia[1], '%H:%M')
                if horafin > horainicio:
                    cantidadHorasLaboradas = horafin-horainicio
                    horasLaboradas =
cantidadHorasLaboradas.total_seconds() / 3600
                    sumaDehoras += horasLaboradas
            fechainicial += timedelta(days=1)

        #Calcular las horas trabajadas en el día actual (Día
)
        if nombres_dias[fechainicial.weekday()] in ansHorarioInfo:
            horaHoy = hoy.strftime('%H:%M')
            horarioDia =
ansHorarioInfo[nombres_dias[hoy.weekday()]].split(";")
            horainicioL = datetime.strptime(horarioDia[0], '%H:%M')

            if auxFechaInicio.date() == hoy.date():
                horainicio = auxFechaInicio.strftime('%H:%M')
                horainicio = max(datetime.strptime(
```

```
                    horainicio, '%H:%M'), horainicioL)
                horafin = datetime.strptime(horaHoy, '%H:%M')

                    sumaDehoras += ((horafin-
horainicio).total_seconds() / 3600)

                else:
                    horafin = datetime.strptime(horaHoy, '%H:%M')
                    sumaDehoras += ((horafin-
horainicioL).total_seconds() / 3600)

            totalhoras = 0
            namePropertyTime = ""
            properties = {}
            if stageInfo.label == "Nuevo":

                if "tiempo_en_etapa_nuevo" in event["inputFields"]:
                    totalhoras = sumaDehoras + \
                        float(event["inputFields"]["tiempo_en_etapa_nue
vo"])
                else:
                    totalhoras = sumaDehoras
                namePropertyTime = "tiempo_en_etapa_nuevo"

            elif stageInfo.label == "Asignado":

                if "tiempo_en_etapa_asignado" in event["inputFields"]:
                    totalhoras = sumaDehoras + \
                        float(event["inputFields"]["tiempo_en_etapa_asi
gnado"])
                else:
                    totalhoras = sumaDehoras
                namePropertyTime = "tiempo_en_etapa_asignado"

            elif stageInfo.label == "En revisión":

                if "tiempo_en_etapa_en_revision" in
event["inputFields"]:
                    totalhoras = sumaDehoras + \
                        float(event["inputFields"]["tiempo_en_etapa_en_
revision"])
                else:
                    totalhoras = sumaDehoras
                namePropertyTime = "tiempo_en_etapa_en_revision"

            elif stageInfo.label == "Información pendiente":

                if "tiempo_en_etapa_informacion_pendiente" in
event["inputFields"]:
                    totalhoras = sumaDehoras + \
                        float(event["inputFields"]
                            ["tiempo_en_etapa_informacion_pendiente"])
                else:
                    totalhoras = sumaDehoras
```

```
                    namePropertyTime =
    "tiempo_en_etapa_informacion_pendiente"

                elif stageInfo.label == "Escalamiento área apoyo":

                    if "tiempo_en_etapa_escalamiento_area_apoyo" in
    event["inputFields"]:
                        totalhoras = sumaDehoras + \
                            float(event["inputFields"]
                                ["tiempo_en_etapa_escalamiento_area_apoyo"]
    )
                    else:
                        totalhoras = sumaDehoras
                    if area_de_escalamiento in event["inputFields"]:
                        tAreaEscalado =
    event["inputFields"][area_de_escalamiento]+sumaDehoras
                    else:
                        tAreaEscalado = sumaDehoras
                    namePropertyTime =
    "tiempo_en_etapa_escalamiento_area_apoyo"
                    properties[area_de_escalamiento] = tAreaEscalado
                    sumaHabil = sumaDehoras

                elif stageInfo.label == "Confirmando cierre":

                    if "tiempo_en_etapa_confirmando_cierre" in
    event["inputFields"]:
                        totalhoras = sumaDehoras + \
                            float(event["inputFields"]
                                ["tiempo_en_etapa_confirmando_cierre"])
                    else:
                        totalhoras = sumaDehoras
                    namePropertyTime = "tiempo_en_etapa_confirmando_cierre"

                elif stageInfo.label == "Cerrado con éxito":

                    if "tiempo_en_etapa_cerrado_con_exito" in
    event["inputFields"]:
                        totalhoras = sumaDehoras + \
                            float(event["inputFields"]
                                ["tiempo_en_etapa_cerrado_con_exito"])
                    else:
                        totalhoras = sumaDehoras
                    namePropertyTime = "tiempo_en_etapa_cerrado_con_exito"

                elif stageInfo.label == "Cerrado sin éxito":

                    if "tiempo_en_etapa_cerrado_sin_exito" in
    event["inputFields"]:
                        totalhoras = sumaDehoras + \
                            float(event["inputFields"]
                                ["tiempo_en_etapa_cerrado_sin_exito"])
                    else:
                        totalhoras = sumaDehoras
                    namePropertyTime = "tiempo_en_etapa_cerrado_sin_exito"
```

```
191.
192.             elif stageInfo.label == "Anulado":
193.
194.                 if "tiempo_en_etapa_anulado" in event["inputFields"]:
195.                     totalhoras = sumaDehoras + \
196.                         float(event["inputFields"]["tiempo_en_etapa_anu
    lado"])
197.                 else:
198.                     totalhoras = sumaDehoras
199.                 namePropertyTime = "tiempo_en_etapa_anulado"
200.             if not namePropertyTime == "":
201.                 properties[namePropertyTime] = totalhoras
202.                 print("properties_update",properties)
203.                 simple_public_object_input = SimplePublicObjectInput(
204.                     properties=properties)
205.                 updateTicket =
    client.crm.tickets.basic_api.update(ticket_id=idTicket,
    simple_public_object_input=simple_public_object_input)
206.
207.         return {"outputFields": {"email": "stage"}}
```

Calculo tiempos ANS cierre (Código personalizado): Este script se encarga de gestionar eventos relacionados con tickets de soporte en HubSpot CRM. Realiza diversas operaciones, incluyendo la extracción de datos del evento, cálculos de tiempo trabajado en el ticket, actualización de propiedades del ticket y evaluación del cumplimiento de acuerdos de nivel de servicio (ANS).

Parámetros:

- `event`: Un diccionario que contiene información sobre el evento que desencadenó la ejecución del script. Este diccionario debe contener los campos necesarios para llevar a cabo las operaciones dentro de la función.

Salidas:

No hay salida explícita de la función, actualiza propiedades del ticket en HubSpot CRM según los cálculos realizados.

Flujo de ejecución:

1. Extracción de Datos del Evento: Se extraen varios campos relevantes del evento, como el ID del ticket, tipo de soporte, etapa del pipeline, etc.
2. Creación del Cliente HubSpot: Se crea un cliente para interactuar con la API de HubSpot CRM.
3. Obtención de Información del Ticket: Se obtiene información sobre el ticket, incluyendo su historial de cambios en la etapa del pipeline.

4. Procesamiento del Ticket: Se realizan cálculos para determinar el tiempo total trabajado en el ticket, evaluación del cumplimiento del ANS, entre otros.
5. Actualización del Ticket: Se actualizan las propiedades del ticket en HubSpot CRM con la nueva información calculada.

Dependencias:

- `os`: Módulo para interactuar con funcionalidades dependientes del sistema operativo, utilizado aquí para acceder a las variables de entorno.
- `hubspot`: Módulo para interactuar con la API de HubSpot CRM.
- `datetime`: Módulo para trabajar con fechas y horas.
- `timedelta`: Clase para representar diferencias de tiempo.
- `timezone`: Clase para representar zonas horarias.

Consideraciones:

- Este script asume que se han configurado las variables de entorno adecuadas para el token de acceso a la API de HubSpot CRM (`IMEXHS_API_KEY`).
- Se espera que el evento proporcionado contenga los campos necesarios para su procesamiento.

```
1. import os
2. import hubspot
3. from hubspot.crm.tickets import ApiException,SimplePublicObjectInput
4. from datetime import datetime,timedelta,timezone
5. 
6. ## ------------------------------------------------------------------
   --##
7. def main(event):
8. 
9.     idTicket = event["inputFields"]["hs_ticket_id"]
10.    tipoSoporte = event["inputFields"]["tipo_soporte"]
11.    pipeline_stage = event["inputFields"]["hs_pipeline_stage"]
12.    pipeline = event["inputFields"]["hs_pipeline"]
13.    hs_ticket_priority = event["inputFields"]["hs_ticket_priority"]
14.    es_distribuidor = "false"
15.    if "es_distribuidor" in event["inputFields"]:
16.        es_distribuidor = event["inputFields"]["es_distribuidor"]
17. 
18.    area_de_escalamiento = ""
19.    if "area_de_escalamiento" in event["inputFields"]:
20.        area_de_escalamiento =
   event["inputFields"]["area_de_escalamiento"]
21. 
22.    dirigidoA = ""
```

```
    if es_distribuidor == "true":
        dirigidoA = "Distribuidor"
    else:
        dirigidoA = "Cliente"

    client = hubspot.Client.create(access_token=os.getenv('ANS_API_KEY'))
    ApiResponse =
client.crm.tickets.basic_api.get_by_id(ticket_id=idTicket, properties=[
                                                                "hs_pipeline_sta
ge"], properties_with_history=["hs_pipeline_stage"], archived=False)
    stage = ApiResponse.properties_with_history.get('hs_pipeline_stage')
    if len(stage) > 1:

        desplazamiento_colombia = timedelta(hours=-5)
        hoy = stage[0].timestamp+desplazamiento_colombia
        fechainicial = stage[1].timestamp+desplazamiento_colombia
        # fechainicialDate=datetime.strptime(fechainicial, "%Y-%m-
%dT%H:%M:%S.%fZ")
        stageInfo = client.crm.pipelines.pipeline_stages_api.get_by_id(
            object_type="tickets", pipeline_id=pipeline,
stage_id=stage[1].value)

        ans = client.cms.hubdb.rows_api.get_table_rows(
            table_id_or_name="15666306")
        ansInfo = {}
        for an in ans.results:
            if an.values["tipo_de_soporte"].lower() ==
tipoSoporte.lower() and dirigidoA.lower() ==
an.values["soporte_dirigido_a"].lower():
                ansInfo = {"id": an.id,
                           "tipo_de_soporte":
an.values["tipo_de_soporte"],
                           "soporte_dirigido_a":
an.values["soporte_dirigido_a"],
                           "th_primera_rta_baja":
an.values["th_primera_rta_baja"],
                           "th_primera_rta_media":
an.values["th_primera_rta_media"],
                           "th_primera_rta_alta":
an.values["th_primera_rta_alta"],
                           "th_cierre_baja": an.values["th_cierre_baja"],
                           "th_cierre_medio":
an.values["th_cierre_medio"],
                           "th_cierre_alta": an.values["th_cierre_alta"]}

        ansHorario = client.cms.hubdb.rows_api.get_table_rows(
            table_id_or_name="15666307")
        ansHorarioInfo = {}
        for ansH in ansHorario.results:
            # print(ansInfo)
            if ansH.values["relacion_ans"][0]["id"] == ansInfo["id"]:
                ansHorarioInfo = ansH.values
        nombres_dias = ["lunes", "martes", "miercoles",
                        "jueves", "viernes", "sabado", "domingo"]
        sumaDehoras = 0
```

```
66.         auxFechaInicio = (stage[1].timestamp+desplazamiento_colombia)
67.         while fechainicial.date() < hoy.date():
68.             if nombres_dias[fechainicial.weekday()] in ansHorarioInfo:
69.                 horarioDia =
    ansHorarioInfo[nombres_dias[fechainicial.weekday()]].split(
70.                     ";")
71.                 if auxFechaInicio.date() == fechainicial.date():
72.                     horainicio = fechainicial.strftime('%H:%M')
73.                     horainicio = max(datetime.strptime(
74.                         horarioDia[0], '%H:%M'),
    datetime.strptime(horainicio, '%H:%M'))
75.                 else:
76.                     horainicio = datetime.strptime(horarioDia[0],
    '%H:%M')
77.                 horafin = datetime.strptime(horarioDia[1], '%H:%M')
78.                 if horafin > horainicio:
79.                     cantidadHorasLaboradas = horafin-horainicio
80.                     horasLaboradas =
    cantidadHorasLaboradas.total_seconds() / 3600
81.                     sumaDehoras += horasLaboradas
82.             fechainicial += timedelta(days=1)
83.
84.         if nombres_dias[fechainicial.weekday()] in ansHorarioInfo:
85.             horaHoy = hoy.strftime('%H:%M')
86.             horarioDia =
    ansHorarioInfo[nombres_dias[hoy.weekday()]].split(";")
87.             horainicioL = datetime.strptime(horarioDia[0], '%H:%M')
88.             if auxFechaInicio.date() == hoy.date():
89.                 horainicio = auxFechaInicio.strftime('%H:%M')
90.                 horainicio = max(datetime.strptime(
91.                     horainicio, '%H:%M'), horainicioL)
92.                 horafin = datetime.strptime(horaHoy, '%H:%M')
93.
94.                 sumaDehoras += ((horafin-horainicio).total_seconds() /
    3600)
95.
96.             else:
97.                 horafin = datetime.strptime(horaHoy, '%H:%M')
98.                 sumaDehoras += ((horafin-horainicioL).total_seconds() /
    3600)
99.
100.             print("sumaDehoras",sumaDehoras)
101.
102.             totalhoras = 0
103.             namePropertyTime = ""
104.             sumaHabil = 0
105.             tAreaEscalado = 0
106.             properties = {}
107.             if stageInfo.label == "Nuevo":
108.
109.                 if "tiempo_en_etapa_nuevo" in event["inputFields"]:
110.                     totalhoras = sumaDehoras + \
111.                         float(event["inputFields"]["tiempo_en_etapa_nue
    vo"])
112.                 else:
```

```
                    totalhoras = sumaDehoras
                namePropertyTime = "tiempo_en_etapa_nuevo"
                sumaHabil = sumaDehoras
            elif stageInfo.label == "Asignado":

                if "tiempo_en_etapa_asignado" in event["inputFields"]:
                    totalhoras = sumaDehoras + \
                        float(event["inputFields"]["tiempo_en_etapa_asi
gnado"])
                else:
                    totalhoras = sumaDehoras
                namePropertyTime = "tiempo_en_etapa_asignado"
                sumaHabil = sumaDehoras
            elif stageInfo.label == "En revisión":

                if "tiempo_en_etapa_en_revision" in
event["inputFields"]:
                    totalhoras = sumaDehoras + \
                        float(event["inputFields"]["tiempo_en_etapa_en_
revision"])
                else:
                    totalhoras = sumaDehoras
                namePropertyTime = "tiempo_en_etapa_en_revision"
                sumaHabil = sumaDehoras
            elif stageInfo.label == "Información pendiente":

                if "tiempo_en_etapa_informacion_pendiente" in
event["inputFields"]:
                    totalhoras = sumaDehoras + \
                        float(event["inputFields"]
                              ["tiempo_en_etapa_informacion_pendiente"]
)
                else:
                    totalhoras = sumaDehoras
                namePropertyTime =
"tiempo_en_etapa_informacion_pendiente"

            elif stageInfo.label == "Escalamiento área apoyo":

                if "tiempo_en_etapa_escalamiento_area_apoyo" in
event["inputFields"]:
                    totalhoras = sumaDehoras + \
                        float(event["inputFields"]
                              ["tiempo_en_etapa_escalamiento_area_apoyo
"])
                else:
                    totalhoras = sumaDehoras
                if area_de_escalamiento in event["inputFields"]:
                    tAreaEscalado =
event["inputFields"][area_de_escalamiento]+sumaDehoras
                else:
                    tAreaEscalado = sumaDehoras
                namePropertyTime =
"tiempo_en_etapa_escalamiento_area_apoyo"
                properties[area_de_escalamiento] = tAreaEscalado
```

```
                    sumaHabil = sumaDehoras

                elif stageInfo.label == "Confirmando cierre":

                    if "tiempo_en_etapa_confirmando_cierre" in
   event["inputFields"]:
                        totalhoras = sumaDehoras + \
                            float(event["inputFields"]
                                  ["tiempo_en_etapa_confirmando_cierre"])
                    else:
                        totalhoras = sumaDehoras
                    namePropertyTime = "tiempo_en_etapa_confirmando_cierre"
                    sumaHabil = sumaDehoras
                elif stageInfo.label == "Cerrado con éxito":

                    if "tiempo_en_etapa_cerrado_con_exito" in
   event["inputFields"]:
                        totalhoras = sumaDehoras + \
                            float(event["inputFields"]
                                  ["tiempo_en_etapa_cerrado_con_exito"])
                    else:
                        totalhoras = sumaDehoras
                    namePropertyTime = "tiempo_en_etapa_cerrado_con_exito"

                elif stageInfo.label == "Cerrado sin éxito":

                    if "tiempo_en_etapa_cerrado_sin_exito" in
   event["inputFields"]:
                        totalhoras = sumaDehoras + \
                            float(event["inputFields"]
                                  ["tiempo_en_etapa_cerrado_sin_exito"])
                    else:
                        totalhoras = sumaDehoras
                    namePropertyTime = "tiempo_en_etapa_cerrado_sin_exito"

                elif stageInfo.label == "Anulado":

                    if "tiempo_en_etapa_anulado" in event["inputFields"]:
                        totalhoras = sumaDehoras + \
                            float(event["inputFields"]["tiempo_en_etapa_anu
   lado"])
                    else:
                        totalhoras = sumaDehoras
                    namePropertyTime = "tiempo_en_etapa_anulado"
                if not namePropertyTime == "":
                    tiempo_en_etapa_nuevo = 0
                    tiempo_en_etapa_anulado = 0
                    tiempo_en_etapa_asignado = 0
                    tiempo_en_etapa_cerrado_con_exito = 0
                    tiempo_en_etapa_cerrado_sin_exito = 0
                    tiempo_en_etapa_confirmando_cierre = 0
                    tiempo_en_etapa_en_revision = 0
                    tiempo_en_etapa_escalamiento_area_apoyo = 0
                    tiempo_en_etapa_informacion_pendiente = 0

```

```
                    if "tiempo_en_etapa_nuevo" in event["inputFields"]:
                        tiempo_en_etapa_nuevo = float(
                            event["inputFields"]["tiempo_en_etapa_nuevo"])

                    if "tiempo_en_etapa_anulado" in event["inputFields"]:
                        tiempo_en_etapa_anulado = float(
                            event["inputFields"]["tiempo_en_etapa_anulado"]
    )

                    if "tiempo_en_etapa_asignado" in event["inputFields"]:
                        tiempo_en_etapa_asignado = float(
                            event["inputFields"]["tiempo_en_etapa_asignado"
    ])

                    if "tiempo_en_etapa_cerrado_con_exito" in
    event["inputFields"]:
                        tiempo_en_etapa_cerrado_con_exito = float(
                            event["inputFields"]["tiempo_en_etapa_cerrado_c
    on_exito"])

                    if "tiempo_en_etapa_cerrado_sin_exito" in
    event["inputFields"]:
                        tiempo_en_etapa_cerrado_sin_exito = float(
                            event["inputFields"]["tiempo_en_etapa_cerrado_s
    in_exito"])

                    if "tiempo_en_etapa_confirmando_cierre" in
    event["inputFields"]:
                        tiempo_en_etapa_confirmando_cierre = float(
                            event["inputFields"]["tiempo_en_etapa_confirman
    do_cierre"])

                    if "tiempo_en_etapa_en_revision" in
    event["inputFields"]:
                        tiempo_en_etapa_en_revision = float(
                            event["inputFields"]["tiempo_en_etapa_en_revisi
    on"])

                    if "tiempo_en_etapa_escalamiento_area_apoyo" in
    event["inputFields"]:
                        tiempo_en_etapa_escalamiento_area_apoyo = float(
                            event["inputFields"]["tiempo_en_etapa_escalamie
    nto_area_apoyo"])

                    if "tiempo_en_etapa_informacion_pendiente" in
    event["inputFields"]:
                        tiempo_en_etapa_informacion_pendiente = float(
                            event["inputFields"]["tiempo_en_etapa_informaci
    on_pendiente"])

                    stageInfoCierre =
    client.crm.pipelines.pipeline_stages_api.get_by_id(
                        object_type="tickets", pipeline_id=pipeline,
    stage_id=pipeline_stage)
                    tiempocierre = 0
```

```
                    propiedadCierre = ""

                    if stageInfoCierre.label == "Cerrado con éxito":
                        propiedadCierre = "tiempo_en_etapa_cerrado_con_exito"
                        tiempocierre = tiempo_en_etapa_nuevo + tiempo_en_etapa_asignado + \
                            tiempo_en_etapa_en_revision + tiempo_en_etapa_escalamiento_area_apoyo
                    elif stageInfoCierre.label == "Cerrado sin éxito":
                        propiedadCierre = "tiempo_en_etapa_cerrado_sin_exito"
                        tiempocierre = tiempo_en_etapa_nuevo + tiempo_en_etapa_asignado + \
                            tiempo_en_etapa_en_revision + tiempo_en_etapa_escalamiento_area_apoyo
                    elif stageInfoCierre.label == "Anulado":
                        propiedadCierre = "tiempo_en_etapa_anulado"
                        tiempocierre = tiempo_en_etapa_nuevo + tiempo_en_etapa_asignado + \
                            tiempo_en_etapa_en_revision + tiempo_en_etapa_escalamiento_area_apoyo
                    cumpleAns = "no cumple"
                    if hs_ticket_priority == "HIGH":
                        horasansInfo = ansInfo["th_cierre_alta"]
                        if tiempocierre <= float(horasansInfo):
                            cumpleAns = "cumple"
                    elif hs_ticket_priority == "MEDIUM":
                        horasansInfo = ansInfo["th_cierre_medio"]
                        if tiempocierre <= float(horasansInfo):
                            cumpleAns = "cumple"
                    elif hs_ticket_priority == "LOW":
                        horasansInfo = ansInfo["th_cierre_baja"]
                        if tiempocierre <= float(horasansInfo):
                            cumpleAns = "cumple"

                    total = tiempo_en_etapa_nuevo + tiempo_en_etapa_anulado + tiempo_en_etapa_asignado + tiempo_en_etapa_cerrado_con_exito + tiempo_en_etapa_cerrado_sin_exito + \
                        tiempo_en_etapa_confirmando_cierre + tiempo_en_etapa_en_revision + \
                        tiempo_en_etapa_escalamiento_area_apoyo + \
                        tiempo_en_etapa_informacion_pendiente + sumaHabil
                    properties[namePropertyTime] = totalhoras
                    properties[propiedadCierre] = tiempocierre
                    properties["ans_cdd"] = cumpleAns
                    # properties={namePropertyTime:totalhoras,propiedadCierre:tiempocierre,"ans":cumpleAns}
                    print("properties_update",properties)
                    simple_public_object_input = SimplePublicObjectInput(
                        properties=properties)
                    updateTicket = client.crm.tickets.basic_api.update(ticket_id=idTicket, simple_public_object_input=simple_public_object_input)
```

```
288.
289.         return {"outputFields": {"email": "stage"}}
```

Copiado de propiedad cerrado “sin éxito” a tmo IMEXHS (Funciones nativas): Actualiza el campo "TMO Etapas IMEXHS (Horas)" con el valor de "Tiempo en etapa cerrada sin éxito (Horas)" cuando se cumplen las siguientes condiciones:

- El valor de "TMO Etapas IMEXHS (Horas)" es desconocido.
- El valor de "Tiempo en etapa cerrada sin éxito (Horas)" es conocido.
- El campo "Pipeline" es "Clientes Directos y Distribuidores".
- El campo "Estado del boleto" es "Cerrado sin éxito".

5. Propiedades:

Para el correcto funcionamiento del desarrollo, se crearon las siguientes propiedades:

Objeto	Nombre	Nombre interno	Tipo de dato	Descripción
Empresa	Autorizada para formulario	autorizada_para_formulario	Casilla de verificación única	Sirve para activar el workflow que permite almacenar la información de la empresa y sus asociaciones dentro de HubDB
Empresa	Es distribuidor?	es_distribuidor	Casilla de verificación única	Sirve para identificar cuando una empresa es distribuidora, y da paso al workflow que permite almacenar solamente las empresas asociadas a esta.
Empresa	Tipo de soporte	tipo_de_soporte	Opciones de lista	Contiene 3 opciones (Gold, Silver, Bronze), las cuales permiten identificar el tipo de soporte de la Empresa.
Empresa	Última modificación sede	ultima_modificacion_sede	Texto	Cuando una sede tiene un ajuste de nombre, un workflow define una fecha en este campo, para que luego otro workflow permita actualizar la empresa.
Sede	Nombre de sede	nombre_de_sede	Texto	Permite definir un nombre a una sede.
Sede	Tipo de soporte	tipo_de_soporte	Opciones de lista	Contiene 3 opciones (Gold, Silver, Bronze), las cuales

				permiten identificar el tipo de soporte de la Sede.
Ticket	Tipo de soporte	tipo_de_soporte	Opciones de lista	Contiene 3 opciones (Gold, Silver, Bronze), las cuales permiten identificar el tipo de soporte de la Sede.

Guía de ejecución

Es importante tener en cuenta que, al inscribir una empresa, ya sea distribuidora o cliente directo, se debe definir primero toda la información y habilitar la propiedad "Autorizada para formulario" al final. Esto ayuda a evitar la sobrecarga del sistema con el uso constante de los Workflows "Actualización HubDB | Empresas" y "Actualización HubDB | Sedes". Se recomienda crear las sedes antes de habilitar esta propiedad.

Dado que esta solución no es nativa de HubSpot, dependemos de las colas disponibles para la ejecución de cualquier workflow.

Empresa Distribuidora:

1. Asociar las empresas correspondientes con la etiqueta "Child Company". (Una empresa distribuidora no puede contener sedes).
   IMPORTANTE: Verifica que la empresa asociada no tenga activada la propiedad "Autorizada para formulario".
2. Definir el tipo de soporte.
3. Seleccionar "Sí" en la propiedad "Es Distribuidor?".
4. Finalmente, seleccionar "Sí" en la propiedad "Autorizada para formulario".

Empresa Cliente Directo:

1. Asociar las sedes correspondientes. Este tipo de empresa no puede tener empresas asociadas con las etiquetas "Parent Company" o "Child Company".
2. Definir el tipo de soporte.
3. Seleccionar "No" en la propiedad "Es Distribuidor?".
4. Finalmente, seleccionar "Sí" en la propiedad "Autorizada para formulario".