Middleware BitMessage para Journey Builder

Documentación técnica completa del middleware que actúa como puente entre Salesforce Marketing Cloud Journey Builder y las APIs de mensajería de BitMessage.

Resumen Ejecutivo

Este middleware es un servidor Express.js que permite a los usuarios de Salesforce Marketing Cloud enviar mensajes SMS a través de las APIs de BitMessage directamente desde sus journeys de automatización. El sistema está diseñado con arquitectura modular para soportar múltiples tipos de actividades de mensajería, tanto instantáneas como programadas (masivas), con capacidad de expansión futura para nuevas funcionalidades.

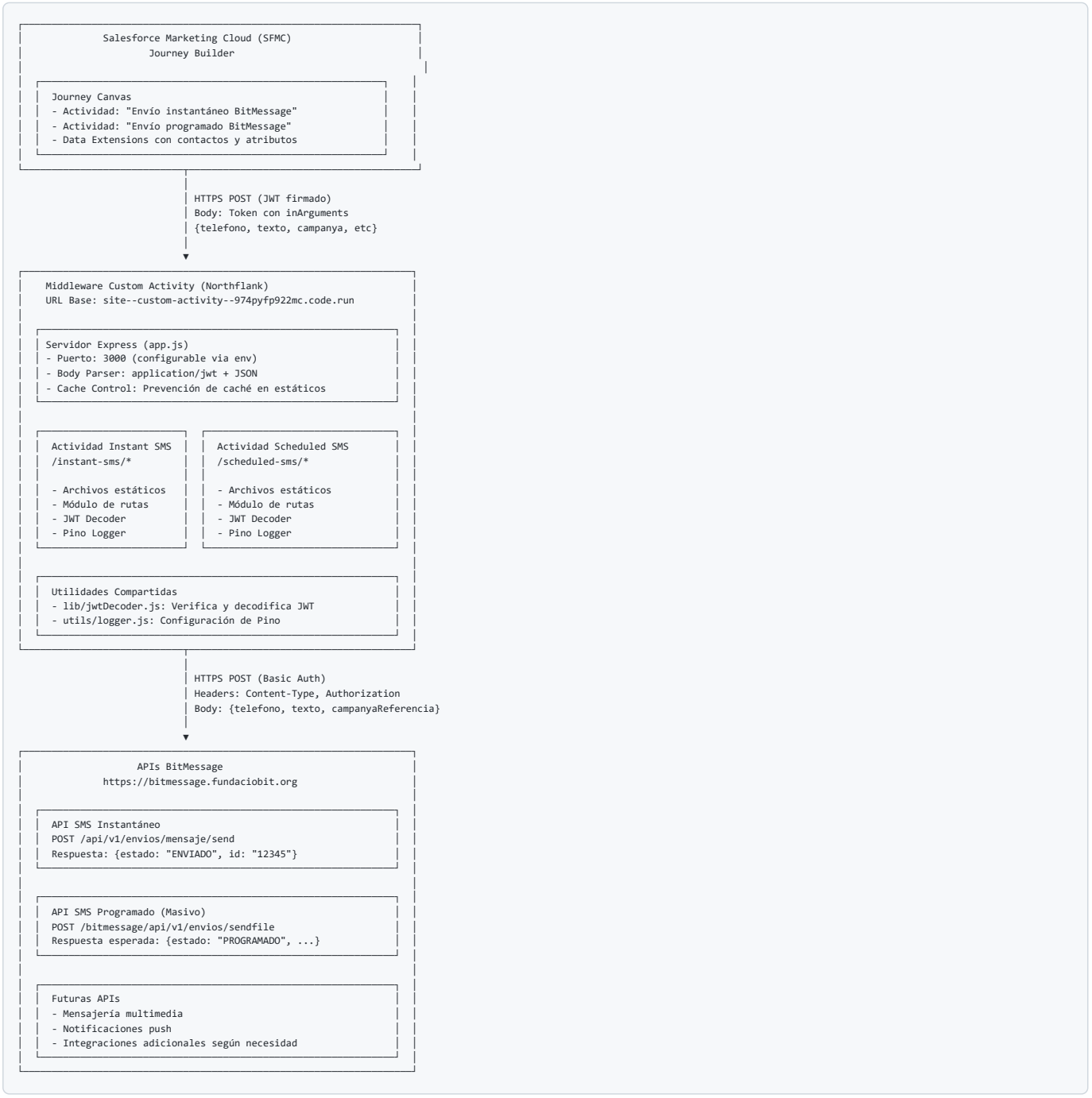
Propósito principal: Actuar como intermediario entre Journey Builder y BitMessage, traduciendo las peticiones de Journey Builder a llamadas válidas de la API de BitMessage y manejando la bifurcación de rutas según el resultado del envío.

Stack tecnológico:

- **Runtime**: Node.js v20+
- **Framework web**: Express v4.15.4
- **Logging**: Pino v10.3.0 + pino-pretty v13.1.3
- **Autenticación**: jsonwebtoken v9.0.2 (JWT con HS256)
- **Cliente HTTP**: Axios v1.8.4
- **Configuración**: dotenv v17.2.3
- **Plataforma**: Northflank (URL: https://site--custom-activity--974pyfp922mc.code.run)

Arquitectura del Sistema

Diagrama de Flujo General



Flujo de Datos Detallado

1. **Configuración en Journey Builder**: El marketer arrastra una actividad al canvas y la configura
2. **Ejecución del Journey**: Cuando un contacto llega a la actividad, Journey Builder envía POST a `/execute`
3. **Autenticación JWT**: El middleware decodifica y valida el token usando el secreto compartido
4. **Extracción de datos**: Se obtienen los parámetros del contacto (teléfono, mensaje, etc.)
5. **Transformación**: Los datos se formatean según la estructura requerida por BitMessage
6. **Llamada a API**: Axios envía POST con Basic Auth a la API correspondiente de BitMessage
7. **Evaluación**: Se analiza la respuesta de BitMessage (estado: ENVIADO, ERROR, etc.)
8. **Bifurcación**: Se devuelve un `branchResult` que indica qué camino debe seguir el contacto
9. **Continuación**: Journey Builder procesa el resultado y dirige al contacto por la rama correspondiente

Estructura del Proyecto

```
bit-middleware/
├── app.js                          # Servidor Express principal
├── package.json                    # Dependencias y configuración npm
├── gulpfile.js                     # Tareas de build (si aplica)
│
├── lib/
│   └── jwtDecoder.js               # Decodificador y verificador de JWT
│
├── utils/
│   └── logger.js                   # Configuración de Pino logger
│
├── routes/
│   └── activities/
│       ├── instant-sms.js          # Endpoints para SMS instantáneo
│       └── scheduled-sms.js        # Endpoints para SMS programado
│
├── public/
│   ├── instant-sms/                # Actividad SMS instantáneo
│   │   ├── config.json             # Configuración Journey Builder
│   │   ├── index.html              # Modal de configuración
│   │   ├── css/
│   │   │   └── styles.css
│   │   ├── js/
│   │   │   ├── customActivity.js  # Lógica frontend (Postmonger)
│   │   │   ├── postmonger.js      # Librería comunicación JB
│   │   │   └── require.js
│   │   └── images/
│   │       ├── icon.png           # 60x60px
│   │       └── iconSmall.png      # 30x30px
│   │
│   └── scheduled-sms/              # Actividad SMS programado
│       ├── config.json
│       ├── index.html
│       ├── css/
│       ├── js/
│       └── images/
│
└── [variables de entorno]
    BITMESSAGE_INSTANT_SMS_API=https://...
    BITMESSAGE_SCHEDULED_SMS_API=https://bitmessage.fundaciobit.org/bitmessage/api/v1/envios/sendfile
    BITMESSAGE_USERNAME=usuario
    BITMESSAGE_PASSWORD=contraseña
    BITMESSAGE_CAMPANYA=SOIB
    jwtSecret=clave_jwt_desde_sfmc
    PORT=3000
```

Patrón de Arquitectura Modular

Cada actividad es completamente autónoma con:

- **Directorio estático propio**: `public/[nombre-actividad]/`
- **Módulo de rutas independiente**: `routes/activities/[nombre-actividad].js`
- **Configuración específica**: `config.json` con endpoints y outcomes únicos
- **Frontend aislado**: HTML, CSS, JS sin dependencias entre actividades
- **URL de registro única**: `https://dominio.com/[nombre-actividad]`

Esta arquitectura permite:

- Agregar nuevas actividades sin modificar las existentes
- Mantener código limpio y sin acoplamientos
- Desplegar actividades independientemente
- Testear y depurar de forma aislada

Configuración de Variables de Entorno

El sistema requiere las siguientes variables de entorno para funcionar:

Variable	Descripción	Ejemplo	Obligatorio
`PORT`	Puerto donde escucha el servidor Express	`3000`	No (default: 3000)
`jwtSecret`	Secret para verificar JWT desde Journey Builder	`abc123def456ghi789`	**Sí**
`BITMESSAGE_USERNAME`	Usuario Basic Auth para APIs BitMessage	`mi_usuario`	**Sí**
`BITMESSAGE_PASSWORD`	Contraseña Basic Auth para APIs BitMessage	`mi_contraseña_segura`	**Sí**
`BITMESSAGE_CAMPANYA`	Referencia de campaña por defecto	`SOIB`	No
`BITMESSAGE_INSTANT_SMS_API`	URL completa API de SMS instantáneo	`https://bitmessage.fundaciobit.org/api/v1/envios/mensaje/send`	**Sí**
`BITMESSAGE_SCHEDULED_SMS_API`	URL completa API de SMS programado (masivo)	`https://bitmessage.fundaciobit.org/bitmessage/api/v1/envios/sendfile`	**Sí**
`NODE_ENV`	Entorno de ejecución (development, production)	`production`	No

Obtención de credenciales

- **jwtSecret**: Se obtiene desde Salesforce Marketing Cloud > Setup > Installed Packages > [Tu Package] > JWT Secret
- **Credenciales BitMessage**: Proporcionadas por el administrador de BitMessage
- **URLs de API**: Documentación de BitMessage o administrador del servicio

Servidor Express (app.js)

Configuración Principal

El archivo principal del servidor configura Express con todos los middleware necesarios y registra las rutas de las actividades:

```
// Load environment variables
import "dotenv/config";

// Module Dependencies
import express from "express";
import bodyParser from "body-parser";
import errorhandler from "errorhandler";
import http from "http";
import path from "path";
import * as instantSms from "./routes/activities/instant-sms.js";
import * as scheduledSms from "./routes/activities/scheduled-sms.js";
import { fileURLToPath } from "url";
import { dirname } from "path";

const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);

var app = express();

// Configure Express
app.set("port", process.env.PORT || 3000);
app.use(bodyParser.raw({ type: "application/jwt" })); // Para JWT de SFMC
app.use(bodyParser.json()); // Para JSON general
app.use(bodyParser.urlencoded({ extended: true })); // Para formularios
```

Puntos clave de la configuración:

- Usa ES6 modules (`type: "module"` en package.json)
- Carga variables de entorno con `dotenv/config`
- Tres parsers diferentes para manejar JWT, JSON y URL-encoded
- Puerto configurable via variable de entorno

Prevención de Caché

Middleware crítico que evita que Journey Builder use versiones cacheadas de archivos estáticos:

```
// Prevent caching of static files
app.use((req, res, next) => {
  res.set("Cache-Control", "no-cache, no-store, must-revalidate");
  res.set("Pragma", "no-cache");
  res.set("Expires", "0");
  next();
});
```

Importancia: Journey Builder cachea agresivamente los archivos estáticos. Sin estos headers, los cambios en config.json, HTML o JS podrían no reflejarse inmediatamente.

Servir Archivos Estáticos

Cada actividad tiene su propio directorio estático independiente:

```
// Serve static files for each activity
app.use(
  "/instant-sms",
  express.static(path.join(__dirname, "public/instant-sms")),
);
app.use(
  "/scheduled-sms",
  express.static(path.join(__dirname, "public/scheduled-sms")),
);
```

Funcionamiento:

- Cuando Journey Builder accede a `https://dominio.com/instant-sms`:
  1. Express sirve automáticamente `public/instant-sms/index.html` como página principal
  2. `config.json` se accede en `/instant-sms/config.json`
  3. Imágenes, CSS y JS se cargan relativos a `/instant-sms/`
- Cada actividad está completamente aislada en su subdirectorio

Registro de Rutas

Todas las actividades exponen 6 endpoints estándar del ciclo de vida de Journey Builder:

```
// ===== Instant SMS Activity Routes =====
app.post("/instant-sms/save", instantSms.save); // Guarda configuración
app.post("/instant-sms/validate", instantSms.validate); // Valida antes de publicar
app.post("/instant-sms/publish", instantSms.publish); // Publica el journey
app.post("/instant-sms/execute", instantSms.execute); // Ejecuta cuando llega contacto
app.post("/instant-sms/stop", instantSms.stop); // Detiene el journey
app.post("/instant-sms/edit", instantSms.edit); // Edita actividad existente

// ===== Scheduled SMS Activity Routes =====
app.post("/scheduled-sms/save", scheduledSms.save);
app.post("/scheduled-sms/validate", scheduledSms.validate);
app.post("/scheduled-sms/publish", scheduledSms.publish);
app.post("/scheduled-sms/execute", scheduledSms.execute);
app.post("/scheduled-sms/stop", scheduledSms.stop);
app.post("/scheduled-sms/edit", scheduledSms.edit);
```

Orden de llamadas típico:

1. **save**: Cada vez que el usuario guarda cambios en el modal
2. **validate**: Antes de publicar el journey (verifica que todo está OK)
3. **publish**: Cuando el journey se activa
4. **execute**: Cada vez que un contacto llega a la actividad (puede ser millones de veces)
5. **stop**: Cuando el journey se desactiva
6. **edit**: Cuando se vuelve a abrir el modal para editar

Inicio del Servidor

```
http.createServer(app).listen(app.get("port"), function () {
  console.log("Express server listening on port " + app.get("port"));
});
```

El servidor se inicia y queda escuchando en el puerto especificado (3000 por defecto).

Decodificador JWT (lib/jwtDecoder.js)

Módulo simple pero crítico que verifica la autenticidad de las peticiones de Journey Builder:

```
import jwt from "jsonwebtoken";

export default (body, secret, cb) => {
  if (!body) {
    return cb(new Error("invalid jwtdata"));
  }

  jwt.verify(body.toString("utf8"), secret, { algorithm: "HS256" }, cb);
};
```

Funcionamiento:

1. Recibe el body raw (tipo `application/jwt`) como Buffer
2. Lo convierte a string UTF-8
3. Verifica la firma usando el secret compartido con SFMC
4. Solo acepta algoritmo HS256 (HMAC con SHA-256)
5. Ejecuta callback con error o datos decodificados

Ejemplo de JWT decodificado:

```
{
  "inArguments": [
    {
      "telefono": "+34600111222",
      "message": "Hola Juan, tu código es: 12345"
    }
  ],
  "outArguments": [],
  "activityObjectID": "abc-123-def",
  "journeyId": "journey-456",
  "activityId": "activity-789",
  "definitionInstanceId": "instance-012",
  "keyValue": "contact@example.com"
}
```

Seguridad: Si el secret no coincide o el token ha sido manipulado, `jwt.verify()` falla y se devuelve HTTP 401 Unauthorized.

Logger Estructurado (utils/logger.js)

Configuración de Pino para logging JSON estructurado:

```
import pino from "pino";

export default pino({
  level: process.env.LOG_LEVEL || "info",
  transport: {
    target: "pino-pretty",
    options: {
      colorize: true,
      translateTime: "SYS:standard",
      ignore: "pid,hostname",
    },
  },
});
```

Características:

- **JSON estructurado**: Cada log es un objeto JSON, fácil de parsear y analizar
- **Niveles configurables**: debug, info, warn, error, fatal
- **Pretty printing**: En desarrollo, los logs se muestran con colores y formato legible
- **Timestamps automáticos**: Cada log incluye timestamp preciso

Niveles de log utilizados:

Nivel	Uso	Ejemplo
`debug`	Información muy detallada (JWT completo, payloads)	`logger.debug({jwt: body}, "JWT received")`
`info`	Operaciones normales (llamadas a endpoints, respuestas)	`logger.info("Execute endpoint called")`
`warn`	Situaciones anómalas pero no errores (estados desconocidos)	`logger.warn({estado}, "Unknown status")`
`error`	Errores que impiden operación (JWT inválido, fallo API)	`logger.error({err}, "API call failed")`

Actividad: SMS Instantáneo

Identificación

- **Key**: `instant-sms-activity-1`
- **Nombre**: "Envío instantáneo BitMessage"
- **Tipo**: `RESTDECISION` (con bifurcación de resultados)
- **URL Base**: `https://site--custom-activity--974pyfp922mc.code.run/instant-sms`

Configuración (public/instant-sms/config.json)

```
{
  "workflowApiVersion": "1.1",
  "metaData": {
    "icon": "images/icon.png",
    "iconSmall": "images/iconSmall.png",
    "category": "custom"
  },
  "type": "RESTDECISION",
  "key": "instant-sms-activity-1",
  "lang": {
    "en-US": {
      "name": "Envío instantáneo BitMessage",
      "description": "Envía SMS vía BitMessage con SFMC",
      "step1Label": "Configure Activity"
    }
  },
  "arguments": {
    "execute": {
      "inArguments": [],
      "outArguments": [],
      "url": "https://site--custom-activity--974pyfp922mc.code.run/instant-sms/execute",
      "verb": "POST",
      "body": "",
      "header": "",
      "format": "json",
      "useJwt": true,
      "timeout": 10000,
      "concurrentRequests": 10
    }
  },
  "configurationArguments": {
    "save": {
      "url": "https://site--custom-activity--974pyfp922mc.code.run/instant-sms/save",
      "verb": "POST",
      "useJwt": true
    },
    "publish": {
      "url": "https://site--custom-activity--974pyfp922mc.code.run/instant-sms/publish",
      "verb": "POST",
      "useJwt": true
    },
    "stop": {
      "url": "https://site--custom-activity--974pyfp922mc.code.run/instant-sms/stop",
      "verb": "POST",
      "useJwt": true
    },
    "validate": {
      "url": "https://site--custom-activity--974pyfp922mc.code.run/instant-sms/validate",
      "verb": "POST",
      "useJwt": true
    },
    "edit": {
      "url": "https://site--custom-activity--974pyfp922mc.code.run/edit",
      "verb": "POST",
      "useJwt": true
    }
  },
  "outcomes": [
    {
      "arguments": {
        "branchResult": "sent"
      },
      "metaData": {
        "label": "Sent"
      }
    },
    {
      "arguments": {
        "branchResult": "notsent"
      },
      "metaData": {
        "label": "Not Sent"
      }
    }
  ],
  "wizardSteps": [{ "label": "Configure Activity", "key": "step1" }],
  "userInterfaces": {
    "configModal": {
      "height": 400,
      "width": 500,
      "fullscreen": false
    }
  },
  "schema": {
    "arguments": {
      "execute": {
        "inArguments": [],
        "outArguments": []
      }
    }
  }
}
```

Puntos clave:

- `useJwt: true` en todos los endpoints - Todas las llamadas están firmadas con JWT
- `timeout: 10000` - 10 segundos máximo por ejecución
- `concurrentRequests: 10` - Hasta 10 contactos procesados simultáneamente
- **Outcomes**: `sent` y `notsent` para bifurcación del journey
- **Modal**: 400x500px, no fullscreen

Frontend (public/instant-sms/js/customActivity.js)

Usa la librería **Postmonger** para comunicarse bidireccionalente con Journey Builder:

```
define(["postmonger"], function (Postmonger) {
  const connection = new Postmonger.Session();
  let authTokens = {};
  let payload = {};

  // DOM element references
  const elements = {
    messageBody: null,
    messageBodyError: null,
    personalize: null,
    prefix: null,
  };

  // Postmonger events subscription
  connection.on("initActivity", initialize);
  connection.on("requestedTokens", onGetTokens);
  connection.on("requestedEndpoints", onGetEndpoints);
  connection.on("requestedInteraction", onRequestedInteraction);
  connection.on(
    "requestedTriggerEventDefinition",
    onRequestedTriggerEventDefinition,
  );
  connection.on("requestedDataSources", onRequestedDataSources);
  connection.on("clickedNext", save);

  // Initialize when DOM is ready
  if (document.readyState === "loading") {
    document.addEventListener("DOMContentLoaded", onRender);
  } else {
    onRender();
  }

  function onRender() {
    // Cache DOM elements
    elements.messageBody = document.getElementById("message-body");
    elements.messageBodyError = document.getElementById("message-body-error");
    elements.personalize = document.getElementById("personalize");
    elements.prefix = document.getElementById("prefix");

    // JB will respond the first time 'ready' is called with 'initActivity'
    connection.trigger("ready");
    connection.trigger("requestTokens");
    connection.trigger("requestEndpoints");
    connection.trigger("requestInteraction");
    connection.trigger("requestTriggerEventDefinition");
    connection.trigger("requestDataSources");
  }

  // This function is called when the custom activity is opened by the user.
  function initialize(data) {
    console.log(data);
    if (data) {
      payload = data;
    }

    // This logic checks if you had previously configured your activity.
    const hasInArguments = Boolean(
      payload.arguments?.execute?.inArguments?.length > 0,
    );

    const inArguments = hasInArguments
      ? payload.arguments.execute.inArguments
      : {};

    console.log(inArguments);

    // For each inArgument, you can pre-populate the fields configured by the user!
    if (hasInArguments) {
      const inArgument = inArguments[0];
      if (elements.messageBody && inArgument.message) {
        elements.messageBody.value = inArgument.message;
      }
    }

    connection.trigger("updateButton", {
      button: "next",
      text: "done",
      visible: true,
    });
  }

  function save() {
    // Here's is where you can validate your attributes before saving the activity
    const messageBodyValue = elements.messageBody?.value?.trim() || "";

    if (messageBodyValue === "") {
      if (elements.messageBodyError) {
        elements.messageBodyError.style.display = "block";
      }
      connection.trigger("ready");
    } else {
      // Hide error message if shown
      if (elements.messageBodyError) {
        elements.messageBodyError.style.display = "none";
      }

      const arg = {
        message: messageBodyValue,
      };

      // This is how you save execute arguments in the activity.
      payload.arguments.execute.inArguments = [arg];
      payload.metaData.isConfigured = true;

      connection.trigger("updateActivity", payload);
    }
  }
});
```

Funcionalidades:

- **Validación**: No permite guardar si el mensaje está vacío
- **Pre-población**: Si la actividad ya estaba configurada, carga el mensaje guardado
- **Data binding**: Soporta sintaxis de Journey Builder como `{{Contact.Attribute.FieldName}}`
- **Vanilla JavaScript**: Sin dependencias externas excepto Postmonger
- **Gestión de DOM**: Cachea referencias a elementos para mejor rendimiento

Backend (routes/activities/instant-sms.js)

Función de Envío a BitMessage

```
/**
 * Sends SMS via BitMessage Instant SMS API
 * @param {Object} payload - SMS payload {telefono, texto, campanyaReferencia}
 * @returns {Promise<{success: boolean, data: Object}>}
 */
async function sendBitMessageSMS(payload) {
  try {
    logger.info({ payload }, "Calling BitMessage Instant SMS API");

    const response = await axios.post(
      process.env.BITMESSAGE_INSTANT_SMS_API,
      payload,
      {
        auth: {
          username: process.env.BITMESSAGE_USERNAME,
          password: process.env.BITMESSAGE_PASSWORD,
        },
        headers: {
          "Content-Type": "application/json",
        },
      },
    );

    logger.info(
      {
        status: response.status,
        estado: response.data?.estado,
        id: response.data?.id,
      },
      "BitMessage Instant SMS API responded",
    );

    const estado = response.data?.estado?.toUpperCase();

    if (estado === "ENVIADO" || estado === "CONFIRMADO") {
      logger.info(
        { smsId: response.data?.id },
        "Instant SMS sent successfully",
      );
      return { success: true, data: response.data };
    } else {
      logger.warn(
        { estado, response: response.data },
        "Unknown BitMessage Instant SMS status",
      );
      return { success: false, data: response.data };
    }
  } catch (error) {
    logger.error(
      { err: error, response: error.response?.data },
      "BitMessage Instant SMS API call failed",
    );
    return { success: false, error };
  }
}
```

Estructura del payload enviado a BitMessage:

```
{
  "telefono": "+34600123456",
  "texto": "Tu mensaje personalizado aquí",
  "campanyaReferencia": "SOIB"
}
```

Respuesta esperada de BitMessage:

```
{
  "estado": "ENVIADO",
  "id": "msg-12345-67890",
  "timestamp": "2026-01-30T10:30:00Z"
}
```

Estados considerados exitosos:

- `ENVIADO` : El mensaje fue enviado exitosamente
- `CONFIRMADO` : El mensaje fue confirmado por la operadora

Cualquier otro estado (ERROR, PENDIENTE, etc.) se considera fallo.

Endpoint Execute

Este es el endpoint más importante, se ejecuta cada vez que un contacto llega a la actividad:

```
export async function execute(req, res) {
  logger.info(
    { endpoint: "/instant-sms/execute" },
    "Instant SMS Execute endpoint called",
  );

  try {
    JWT(req.body, process.env.jwtSecret, async (err, decoded) => {
      logger.debug({ jwt: req.body.toString("utf8") }, "JWT received");

      // verification error -> unauthorized request
      if (err) {
        logger.error({ err }, "JWT verification failed");
        return res.status(401).end();
      }

      if (decoded && decoded.inArguments && decoded.inArguments.length > 0) {
        const decodedArgs = decoded.inArguments[0];
        logger.info({ args: decodedArgs }, "Decoded inArguments");

        // Prepare BitMessage payload
        const smsPayload = {
          telefono: decodedArgs.telefono || decodedArgs.phone,
          texto: decodedArgs.texto || decodedArgs.message,
          campanyaReferencia: process.env.BITMESSAGE_CAMPANYA || "SOIB",
        };

        // Send SMS via BitMessage API
        const result = await sendBitMessageSMS(smsPayload);

        if (result.success) {
          logger.info({ branchResult: "sent" }, "Returning success branch");
          return res.status(200).json({ branchResult: "sent" });
        } else {
          logger.warn({ branchResult: "notsent" }, "Returning failure branch");
          return res.status(200).json({ branchResult: "notsent" });
        }
      } else {
        logger.warn("Invalid inArguments");
        return res.status(200).json({ branchResult: "notsent" });
      }
    });
  } catch (err) {
    logger.error({ err }, "Execute endpoint error");
    return res.status(200).json({ branchResult: "notsent" });
  }
}
```

Flujo de ejecución:

1. Journey Builder envía POST con JWT cuando contacto llega a actividad
2. Se verifica JWT con el secreto compartido:
   - Si falla → HTTP 401 Unauthorized
   - Si OK → continúa
3. Se extraen `inArguments[0]` que contiene los datos del contacto
4. Se construye el payload para BitMessage:
   - `telefono`: acepta alias `phone` para flexibilidad
   - `texto`: acepta alias `message`
   - `campanyaReferencia`: usa variable de entorno o “SOIB” por defecto
5. Axios hace POST con Basic Auth a BitMessage
6. Se evalúa el campo `estado` de la respuesta:
   - “ENVIADO” o “CONFIRMADO” → éxito
   - Cualquier otro valor → fallo
7. Se devuelve siempre HTTP 200 con JSON `{branchResult: "sent"}` o `{branchResult: "notsent"}`
8. Journey Builder lee el `branchResult` y dirige contacto por la rama correspondiente

Importante: El éxito o fallo se comunica mediante `branchResult`, no mediante el código HTTP. Siempre se devuelve 200 OK.

Otros Endpoints del Ciclo de Vida

```
export async function save(req, res) {
  logger.info(
    { endpoint: "/instant-sms/save", body: req.body },
    "Instant SMS Save event received",
  );
  res.send(200, "Save");
}

export async function validate(req, res) {
  logger.info(
    { endpoint: "/instant-sms/validate", body: req.body },
    "Instant SMS Validate event received",
  );
  res.send(200, "Validate");
}

export async function publish(req, res) {
  logger.info(
    { endpoint: "/instant-sms/publish", body: req.body },
    "Instant SMS Publish event received",
  );
  res.send(200, "Publish");
}

export async function stop(req, res) {
  logger.info(
    { endpoint: "/instant-sms/stop", body: req.body },
    "Instant SMS Stop event received",
  );
  res.send(200, "Stop");
}

export async function edit(req, res) {
  logger.info(
    { endpoint: "/instant-sms/edit", body: req.body },
    "Instant SMS Edit event received",
  );
  res.send(200, "Edit");
}
```

Actualmente estos endpoints solo registran el evento para auditoría. Pueden extenderse en el futuro para:

- **save**: Validar configuración o guardar en base de datos
- **validate**: Verificar credenciales antes de publicar
- **publish**: Notificar a sistemas externos
- **stop**: Limpiar recursos o cancelar envíos pendientes
- **edit**: Cargar configuración previa desde BD

Uso en Journey Builder

Pasos para configurar:

1. Arrastrar actividad "Envío instantáneo BitMessage" al canvas
2. Conectar campo de teléfono desde Data Extension: `{{Event.DEName.Phone}}`
3. Escribir mensaje en el modal (puede incluir personalización):

   ```
   Hola {{Contact.Attribute.FirstName}}, tu código es: {{Event.DEName.VerificationCode}}
   ```

4. Hacer clic en "Done" para guardar
5. Conectar outcomes:
   - **Sent** → Wait 1 hour → Engagement Split
   - **Not Sent** → Send Email → Wait → Decision Split

Ejemplo de flujo completo:

```
[Data Extension Entry]
         ↓
[Decision Split: Phone exists?]
    ↓ Yes              ↓ No
[SMS Instantáneo]   [Update Contact]
    ↓ Sent   ↓ Not Sent
[Wait 1h]  [Send Email]
    ↓           ↓
[Goal Check] [Wait 2h]
              ↓
         [End Journey]
```

Actividad: SMS Programado (Masivo)

Identificación

- **Key**: `scheduled-sms-activity-1`
- **Nombre**: "Envío programado BitMessage"
- **Tipo**: `RESTDECISION`
- **URL Base**: `https://site--custom-activity--974pyfp922mc.code.run/scheduled-sms`

Configuración (public/scheduled-sms/config.json)

Similar a la actividad instantánea, con estas diferencias clave:

```
{
  "key": "scheduled-sms-activity-1",
  "lang": {
    "en-US": {
      "name": "Envío programado BitMessage",
      "description": "Programa SMS para envío posterior vía BitMessage",
      "step1Label": "Configure Scheduled SMS"
    }
  },
  "arguments": {
    "execute": {
      "url": "https://site--custom-activity--974pyfp922mc.code.run/scheduled-sms/execute",
      ...
    }
  },
  "outcomes": [
    {
      "arguments": {"branchResult": "scheduled"},
      "metaData": {"label": "Scheduled"}
    },
    {
      "arguments": {"branchResult": "failed"},
      "metaData": {"label": "Failed"}
    }
  ]
}
```

Diferencias principales:

- **Outcomes**: `scheduled` / `failed` en lugar de `sent` / `notsent`
- **URLs**: Todos los endpoints usan `/scheduled-sms/` en lugar de `/instant-sms/`

Frontend (public/scheduled-sms/js/customActivity.js)

Incluye validación de campo adicional de referencia de campaña:

```
function save() {
  const campaignValue = elements.campaignReference?.value?.trim() || "";
  const messageBodyValue = elements.messageBody?.value?.trim() || "";
  let isValid = true;

  // Validar campaña (obligatorio)
  if (!campaignValue) {
    if (elements.campaignError) {
      elements.campaignError.style.display = "block";
    }
    isValid = false;
  } else if (elements.campaignError) {
    elements.campaignError.style.display = "none";
  }

  // Validar mensaje (obligatorio)
  if (!messageBodyValue) {
    if (elements.messageBodyError) {
      elements.messageBodyError.style.display = "block";
    }
    isValid = false;
  } else if (elements.messageBodyError) {
    elements.messageBodyError.style.display = "none";
  }

  if (!isValid) {
    connection.trigger("ready");
  } else {
    // Guardar ambos campos
    payload.arguments.execute.inArguments = [
      {
        campanyaReferencia: campaignValue,
        message: messageBodyValue,
      },
    ];
    payload.metaData.isConfigured = true;
    connection.trigger("updateActivity", payload);
  }
}
```

Campos configurables:

1. **Referencia de campaña** (obligatorio):
    - Input de texto libre configurado por el usuario
    - Identifica la campaña masiva para trazabilidad
    - Se envía a BitMessage como `campanyaReferencia`
    - Ejemplo: “PROMO_VERANO_2026”, “NOTIF_MANTENIMIENTO_01”
2. **Mensaje** (obligatorio):
    - Textarea con soporte de data binding
    - Mismo comportamiento que en actividad instantánea
3. **Fecha/hora de envío** (pendiente):
    - Todavía no implementado
    - Se agregará date/time picker
    - Se enviará como `fechaEnvio` a BitMessage

Backend (routes/activities/scheduled-sms.js)

```
/**
 * Sends Scheduled SMS via BitMessage API
 * @param {Object} payload - SMS payload with scheduling parameters
 * @returns {Promise<{success: boolean, data: Object}>}
 */
async function sendScheduledSMS(payload) {
  try {
    logger.info({ payload }, "Calling BitMessage Scheduled SMS API");

    const response = await axios.post(
      process.env.BITMESSAGE_SCHEDULED_SMS_API,
      payload,
      {
        auth: {
          username: process.env.BITMESSAGE_USERNAME,
          password: process.env.BITMESSAGE_PASSWORD,
        },
        headers: {
          "Content-Type": "application/json",
        },
      },
    );

    const estado = response.data?.estado?.toUpperCase();

    // TODO: Ajustar según respuesta real de la API
    if (estado === "PROGRAMADO" || estado === "ENVIADO") {
      logger.info(
        { smsId: response.data?.id },
        "Scheduled SMS created successfully",
      );
      return { success: true, data: response.data };
    } else {
      logger.warn({ estado, response: response.data }, "Unknown status");
      return { success: false, data: response.data };
    }
  } catch (error) {
    logger.error(
      { err: error, response: error.response?.data },
      "API call failed",
    );
    return { success: false, error };
  }
}

export async function execute(req, res) {
  logger.info(
    { endpoint: "/scheduled-sms/execute" },
    "Execute endpoint called",
  );

  try {
    JWT(req.body, process.env.jwtSecret, async (err, decoded) => {
      if (err) {
        logger.error({ err }, "JWT verification failed");
        return res.status(401).end();
      }

      if (decoded && decoded.inArguments && decoded.inArguments.length > 0) {
        const decodedArgs = decoded.inArguments[0];
        logger.info({ args: decodedArgs }, "Decoded inArguments");

        // TODO: Ajustar payload según estructura de API de SMS programados
        const smsPayload = {
          telefono: decodedArgs.telefono || decodedArgs.phone,
          texto: decodedArgs.texto || decodedArgs.message,
          fechaEnvio: decodedArgs.fechaEnvio || decodedArgs.scheduledDate,
          campanyaReferencia:
            decodedArgs.campanyaReferencia ||
            process.env.BITMESSAGE_CAMPANYA ||
            "SOIB",
        };

        const result = await sendScheduledSMS(smsPayload);

        if (result.success) {
          logger.info(
            { branchResult: "scheduled" },
            "Returning success branch",
          );
          return res.status(200).json({ branchResult: "scheduled" });
        } else {
          logger.warn({ branchResult: "failed" }, "Returning failure branch");
          return res.status(200).json({ branchResult: "failed" });
        }
      } else {
        logger.warn("Invalid inArguments");
        return res.status(200).json({ branchResult: "failed" });
      }
    });
  } catch (err) {
    logger.error({ err }, "Execute endpoint error");
    return res.status(200).json({ branchResult: "failed" });
  }
}
```

Estado actual:

- ✅ Estructura del código completa
- ✅ Campo de campaña implementado y validado
- ✅ Logging y manejo de errores
- ✅ Variable `BITMESSAGE_SCHEDULED_SMS_API` definida: `https://bitmessage.fundaciobit.org/bitmessage/api/v1/envios/sendfile`
- ⚠️ Estados de respuesta (`PROGRAMADO` , etc.) por confirmar según API real
- ⚠️ Campo de fecha/hora pendiente en frontend

Tabla Comparativa de Actividades

Característica	SMS Instantáneo	SMS Programado (Masivo)
URL de registro	`/instant-sms`	`/scheduled-sms`
Key del config	`instant-sms-activity-1`	`scheduled-sms-activity-1`
Nombre visible	"Envío instantáneo BitMessage"	"Envío programado BitMessage"
Campos en modal	Mensaje	Campaña, Mensaje, (Fecha pendiente)
Validación frontend	Mensaje obligatorio	Campaña y Mensaje obligatorios
Parámetros a BitMessage	telefono, texto, campanyaReferencia (env)	telefono, texto, fechaEnvio, campanyaReferencia (user)
API BitMessage	`https://bitmessage.fundaciobit.org/.../mensaje/send`	`https://bitmessage.fundaciobit.org/bitmessage/api/v1/envios/sendfile`
Estados exitosos	ENVIADO, CONFIRMADO	PROGRAMADO, ENVIADO (por confirmar)
Outcome éxito	`sent`	`scheduled`
Outcome fallo	`notsent`	`failed`
Timeout	10 segundos	10 segundos
Concurrencia	10 contactos simultáneos	10 contactos simultáneos
Uso típico	OTPs, confirmaciones, alertas tiempo real	Campañas publicitarias, recordatorios masivos
Estado implementación	Completo y funcional	Estructura completa, API pendiente

Seguridad y Autenticación

Capas de Seguridad

El sistema implementa múltiples capas de autenticación:

1. HTTPS/TLS:

- Certificado SSL en Northflank
- Encriptación de datos en tránsito
- Previene ataques man-in-the-middle

2. JWT de Journey Builder:

- Token firmado con algoritmo HS256
- Secret compartido solo entre SFMC y middleware
- Verifica que la petición viene de Journey Builder legítimo
- Incluye datos del contacto y contexto del journey

3. Validación de datos:

- Verificación de estructura de `inArguments`
- Sanitización de campos (teléfono, mensaje)
- Prevención de injection attacks

4. Basic Auth a BitMessage:

- Credenciales almacenadas en variables de entorno
- No expuestas en código ni logs
- Autenticación específica para cada API call

Almacenamiento de Credenciales

Buenas prácticas implementadas:

- ✅ Credenciales solo en variables de entorno
- ✅ Nunca en código fuente
- ✅ Nunca en repositorio Git
- ✅ Acceso limitado en plataforma de deployment
- ❌ No hacer commit de archivos `.env`
- ❌ No logear valores completos de credenciales
- ❌ No enviar credenciales por canales inseguros

Variables críticas:

```
jwtSecret=abc123def456...                    # Desde SFMC
BITMESSAGE_USERNAME=usuario_produccion       # Desde BitMessage
BITMESSAGE_PASSWORD=password_segura          # Desde BitMessage
```

Protección contra Ataques

JWT Replay Attacks:

- Journey Builder incluye `activityObjectID` único por ejecución
- Aunque alguien capture un JWT, el contacto ya habrá procesado
- BitMessage puede implementar deduplicación por ID de mensaje

Injection Attacks:

- Los datos se envían como JSON, no se construyen queries SQL
- Axios escapa automáticamente caracteres especiales
- Validación de entrada en frontend y backend

Rate Limiting:

- Journey Builder controla: `concurrentRequests: 10`
- BitMessage puede tener límites adicionales
- Pino logger detecta patrones anómalos

DDoS Protection:

- Northflank proporciona protección en capa de red
- URLs de endpoints no son públicas
- Solo Journey Builder conoce las URLs

Logging y Monitoreo

Estructura de Logs

Ejemplo de secuencia de logs para un envío exitoso:

```
{
  "level": 30,
  "time": 1706612400000,
  "endpoint": "/instant-sms/execute",
  "msg": "Instant SMS Execute endpoint called"
}

{
  "level": 30,
  "time": 1706612401000,
  "args": {
    "telefono": "+34600111222",
    "message": "Tu pedido está en camino"
  },
  "msg": "Decoded inArguments"
}

{
  "level": 30,
  "time": 1706612402000,
  "payload": {
    "telefono": "+34600111222",
    "texto": "Tu pedido está en camino",
    "campanyaReferencia": "SOIB"
  },
  "msg": "Calling BitMessage Instant SMS API"
}

{
  "level": 30,
  "time": 1706612403000,
  "status": 200,
  "estado": "ENVIADO",
  "id": "msg-12345",
  "msg": "BitMessage Instant SMS API responded"
}

{
  "level": 30,
  "time": 1706612403500,
  "branchResult": "sent",
  "msg": "Returning success branch"
}
```


Búsqueda y Análisis

Ver ejecuciones de SMS instantáneo:

```
cat logs.json | grep '"endpoint":"/instant-sms/execute"'
```

Encontrar envíos fallidos:

```
cat logs.json | grep '"branchResult":"notsent"'
```

Identificar errores de JWT:

```
cat logs.json | grep '"level":50' | grep "JWT"
```

Calcular tasa de éxito:

```
# Contar envíos exitosos
cat logs.json | grep branchResult | grep -c "sent"
# Comparar con total de executes
```

Alertas Recomendadas

1. **Tasa de error > 10%**: Ratio `notsent / total` supera umbral
2. **JWT verification failures**: Múltiples errores 401 en corto período
3. **Timeouts frecuentes**: Muchos "API call failed" con timeout
4. **Estados desconocidos**: Logs `warn` con "Unknown status"

Despliegue en Northflank

Configuración de Variables

En Northflank > Settings > Environment Variables:

```
jwtSecret = [valor desde SFMC]
BITMESSAGE_USERNAME = [usuario BitMessage]
BITMESSAGE_PASSWORD = [contraseña BitMessage]
BITMESSAGE_CAMPANYA = SOIB
BITMESSAGE_INSTANT_SMS_API = https://bitmessage.fundaciobit.org/api/v1/envios/mensaje/send
BITMESSAGE_SCHEDULED_SMS_API = https://bitmessage.fundaciobit.org/bitmessage/api/v1/envios/sendfile
NODE_ENV = production
LOG_LEVEL = info
```

Proceso de Deployment

1. **Push a repositorio Git**
2. **Northflank detecta cambios** via webhook
3. **Build automático**: `npm install` + `npm run postinstall`
4. **Start**: `npm start` (ejecuta `node app.js`)
5. **Health check**: Verifica puerto 3000 responde
6. **Logs disponibles**: Panel de Northflank o exportación externa

Escalado

Si el volumen aumenta:

- Aumentar `concurrentRequests` en config.json
- Escalar horizontalmente (más instancias)
- Escalar verticalmente (más CPU/RAM)
- Northflank hace load balancing automático

Extensión del Sistema

Agregar Nueva Actividad

Pasos para agregar una nueva actividad (ejemplo: WhatsApp):

1. **Crear estructura**:

```
mkdir -p public/whatsapp-messages/{css,js,images}
touch routes/activities/whatsapp-messages.js
```

2. **Crear config.json** con key único y endpoints
3. **Desarrollar frontend** (copiar y adaptar customActivity.js)
4. **Implementar backend**:
    - Función de llamada a API
    - Endpoint execute con lógica específica
    - Endpoints del ciclo de vida
5. **Registrar en app.js**:

```
import * as whatsappMessages from "./routes/activities/whatsapp-messages.js";
app.use("/whatsapp-messages", express.static(...));
app.post("/whatsapp-messages/execute", whatsappMessages.execute);
// ... otros endpoints
```

6. **Agregar variable de entorno**: `BITMESSAGE_WHATSAPP_API`
7. **Desplegar y registrar** en Journey Builder

Resolución de Problemas

JWT verification failed

Síntoma: Logs con error 401 “JWT verification failed”

Causas:

1. Secret de JWT incorrecto
2. SFMC regeneró el secret
3. Petición sin JWT válido

Solución:

```
# Verificar secret en SFMC
# Comparar con variable de entorno
# Actualizar en Northflank si difiere
# Reiniciar aplicación
```

SMS no se envía (estado "notsent")

Síntoma: Branch result siempre "notsent"

Diagnóstico:

```
cat logs.json | grep "BitMessage.*responded" | tail -n 20
# Revisar campo "estado" en respuesta
```

Causas:

1. BitMessage devuelve estado no contemplado
2. Campo `estado` tiene formato diferente
3. Error de red

Solución: Ajustar lógica de estados en el código

Actividad no aparece en Journey Builder

Checklist:

- [ ] URL accesible públicamente
- [ ] config.json válido (JSONLint)
- [ ] Endpoints correctos
- [ ] Iconos existen
- [ ] SFMC tiene acceso
- [ ] Category en minúsculas

Test:

```
curl https://site--custom-activity--974pyfp922mc.code.run/instant-sms/config.json
```

Timeout en ejecución

Causas:

1. BitMessage API lenta (> 10s)
2. Problema de red
3. Payload muy grande

Soluciones:

- Aumentar timeout en config.json
- Optimizar payload
- Implementar retry logic

Conclusiones

Este middleware proporciona una solución robusta y extensible para integrar Journey Builder con BitMessage. La arquitectura modular permite agregar nuevas actividades sin modificar código existente.

Puntos Fuertes

✅ Arquitectura modular independiente por actividad ✅ Seguridad multi-capa (JWT + Basic Auth + HTTPS) ✅ Logging completo con Pino JSON estructurado ✅ Manejo de errores con bifurcación de rutas ✅ Documentación completa de código y configuraciones ✅ Despliegue automatizado con Northflank

Áreas de Mejora

⚠️ Completar implementación de SMS Programado ⚠️ Agregar suite de tests automatizados ⚠️ Considerar persistencia en BD para configuraciones ⚠️ Implementar dashboard de métricas ⚠️ Mejorar validación de formatos de teléfono

Próximos Pasos

1. Confirmar URL de API de SMS programados con BitMessage
2. Implementar date/time picker en scheduled-sms
3. Testing completo con datos reales
4. Considerar nuevas actividades según necesidades
5. Dashboard de monitoreo si el volumen lo justifica

Versión: 2.0.0
Última actualización: 30 de enero de 2026
Plataforma: Northflank - https://site--custom-activity--974pyfp922mc.code.run