# MODELO FUNCIONAL PARA EL MONITOREO CARTOGRÁFICO DE LAS CONDICIONES METEOROLÓGICAS Y LAS ESPECIFICACIONES TÉCNICAS DE UN VUELO SIMULADO.

**JULIAN RICARDO VILLABONA ARIÓN**
**CRISTIAN LEONARDO AMADO JAIMES**

**UNIVERSIDADINDUSTRIALDESANTANDER**
**FACULTADDEINGENIERÍASFÍSICO-MECÁNICAS**
**ESCUELADEINGENIERIADESISTEMASEINFORMÁTICA**
**BUCARAMANGA**
**2017**

# MODELO FUNCIONAL PARA EL MONITOREO CARTOGRÁFICO DE LAS CONDICIONES METEOROLÓGICAS Y LAS ESPECIFICACIONES TÉCNICAS DE UN VUELO SIMULADO

**JULIAN RICARDO VILLABONA ARIÓN**
**CRISTIAN LEONARDO AMADO JAIMES**

**Trabajo de Grado para optar al título de**
**Ingeniero de Sistemas**

**Director:**
**FERNANDO ANTONIO ROJAS MORALES**
**M.Sc. Ingeniero de Sistemas**

**UNIVERSIDAD INDUSTRIAL DE SANTANDER**
**FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS**
**ESCUELA DE INGENIERIA DE SISTEMAS E INFORMÁTICA**
**BUCARAMANGA**
**2017**

**DEDICATORIA**

**El autor Julian Ricardo Villabona Arión dedicó este logro a:**

A todos

## DEDICATORIA

**El autor Cristian Leonardo Amado Jaimes dedicó este logro a:**

*A todos*

## AGRADECIMIENTOS

A la Universidad Industrial de Santander por la disposición de sus instalaciones y proporcionar un ecosistema apto para el desarrollo profesional.

A la escuela de Ingeniería de Sistemas e Informática y cada uno de sus profesores que con su responsabilidad y entrega aportaron una sólida e integral formación durante nuestra formación personal y profesional.

Al profesor Fernando Antonio Rojas, por la oportunidad de participar en este proyecto y siendo un gran líder durante todo el proceso y permitiéndonos desarrollar habilidades necesarias para enfrentarnos a un mundo laboral.

**CONTENIDO**

## LISTA DE FIGURAS

Pág.

## LISTA DE ANEXOS

## RESUMEN

**TÍTULO:** MODELO FUNCIONAL PARA EL MONITOREO CARTOGRÁFICO DE LAS CONDICIONES METEOROLÓGICAS Y LAS ESPECIFICACIONES TÉCNICAS DE UN VUELO SIMULADO. ([1]*)

**AUTOR:** CRISTIAN LEONARDO AMADO JAIMES ([2]**)
JULIAN RICARDO VILLABONA ARIÓN

**PALABRAS CLAVE:** SOFTWARE AERONÁUTICO, SISTEMA DE INFORMACIÓN, MONITOREO DE VUELOS, AERODROMO, PILOTO, AERONAVE.

**DESCRIPCIÓN:**

Los aviones normalmente se comunican con las estaciones terrestres usando varios sistemas. Ubicados estratégicamente, se encuentra una constelación de satélites transmitiendo continuamente señales dentro de rangos establecidos, cuya función consiste en localizar y posicionar dispositivos en cualquier parte de la tierra. En la aviación, sin embargo, la información sobre la posición geográfica, se transmite mediante transpondedores (dispositivo transmisor / contestador) y comunicaciones de voz y/o de datos por radio. Esto es posible gracias al sistema de comunicación codificada por radio ACARS (Sistema de Reportes y Direccionamiento de Comunicaciones Aeronáuticas), que permite a las aerolíneas vigilar el estado de cada aeronave en vuelo, el control automático del estado del avión, así como el encaminamiento de comunicaciones operativas y logísticas. Siempre será muy conveniente que los pilotos puedan recibir la mayor cantidad de información posible desde tierra, respecto a las condiciones atmosféricas o datos técnicos del vuelo; así como es necesario centralizar los datos y disponer de instrumentos visuales que permitan tomar decisiones rápidas en las centrales aeronáuticas. Esta información debe ser proporcionada por sistemas informáticos localizados en los centros de control operacional de las aerolíneas, los cuales deben ser capaces de interactuar con los dispositivos actualmente utilizados bajo los protocolos de ACARS. Para ello se ha desarrollado un modelo funcional (o prototipo) para el monitoreo cartográfico de las condiciones meteorológicas y las especificaciones técnicas de un vuelo simulado.


[1](*) Trabajo de Grado

[2](**) Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Fernando Antonio Rojas Morales, M.Sc. Ingeniería de Sistemas.

## ABSTRACT

**TITLE:** FUNCTIONAL MODEL FOR CARTOGRAPHIC MONITORING OF WEATHER CONDITIONES AND TECHNICAL SPECIFICATIONS OF A SIMULATED FLIGHT. ([3]*)

**AUTHOR:** CRISTIAN LEONARDO AMADO JAIMES ([4]**)
JULIAN RICARDO VILLABONA ARIÓN

**KEYWORDS:** AERONAUTIC SOFTWARE, INFORMATION SYSTEMS, FLIGHT TRACKIG, AERODROME, PILOT, AIRCRAFT.

**DESCRIPTION:**

Airplanes usually communicate with land stations using various systems. Strategically located, there are a constellation of satellites continuously transmitting signals within established ranges, whose function is to locate and position devices in any part of the earth. In aviation, however, information on geographic position is transmitted through transponders (transmitter / answering device), and voice and/or radio data communications. This is possible thanks to the radio-coded communication system ACARS (Aircraft Communication Addressing and Reporting System), which allows airlines to monitor the status of each aircraft in flight, automatic control of aircraft status, and routing of operational communications And logistics. It will always be very convenient for pilots to receive as much information as possible from the ground, in relation to atmospheric conditions or technical data of the flight; as well as it is necessary to centralize the data and to have visual instruments that allow to make fast decisions in the aeronautical plants. This information must be provided by computer systems located at the operational control centers of the airlines, which must be capable of interacting with the devices currently used under the ACARS protocols. For this purpose a functional model (or prototype) has been developed for the cartographic monitoring of the meteorological conditions and the technical specifications of a simulated flight.


[3](*) Bachelor Thesis.
[4](**) Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director. Fernando Antonio Rojas Morales, M.Sc. Ingeniería de Sistemas.

# INTRODUCCIÓN

Los aviones normalmente se comunican con las estaciones terrestres usando varios sistemas. El control de tráfico aéreo combina las propiedades de los radares básicos para medir ubicaciones, con las señales que proveen los transpondedores (una abreviatura de "transmisor respondedor") de las aeronaves. Con ambos datos se produce una imagen detallada del tráfico en los cielos. Todos los aviones comerciales están equipados con transpondedores que automáticamente transmiten señales electrónicas de vuelta a tierra cuando recibe una señal de radio.

Ubicados estratégicamente, se encuentra una constelación de satélites transmitiendo continuamente señales dentro de rangos establecidos, cuya función consiste en localizar y posicionar dispositivos en cualquier parte de la tierra. De esta manera surge la oportunidad de utilizar tal tecnología, para que mediante un sistema de información bien estructurado, se pueda hacer seguimiento a los vuelos comerciales, manteniendo una comunicación discreta con cada uno de ellos y automatizando algunas tareas.

# 1 DESCRIPCIÓN DEL PROYECTO

## 1.1 DEFINICIÓN DEL PROBLEMA

Hoy por hoy la ubicación satelital es un recurso con el que cuenta un gran número de dispositivos electrónicos. En la aviación, sin embargo, la información sobre la posición geográfica, se transmite mediante transpondedores (señal automática de radiofrecuencia) y comunicaciones de voz y/o de datos por radio, las últimas legislaciones en materia de regulación aeronáutica exigen que las aerolíneas tengan conocimiento en todo momento de la posición geográfica y trayectoria de vuelo de sus aeronaves.

## 1.2 JUSTIFICACIÓN

El clima juega un papel importante, aunque no definitivo en las causas de los accidentes aéreos. La mayoría de los aviones tienen radares meteorológicos y sistemas sofisticados, con lo cual pueden evitar las zonas más complicadas de tormenta, pero estos sistemas tienen rangos de operación limitados y en ocasiones el piloto no posee la visualización más completa del fenómeno atmosférico al que está a punto de enfrentar.

Siempre será muy conveniente que los pilotos puedan recibir la mayor cantidad de información posible desde tierra, respecto a condiciones atmosféricas adversas que pueden ser proporcionadas por sistemas de información localizados en los centros de control operacional de las aerolíneas. Bien sea para disminuir las demoras al vuelo por la presencia de mal tiempo, para no sufrir de esas molestas turbulencias que causan temor a buena parte de los pasajeros, como también para aumentar la seguridad operacional evitando, con la debida anticipación, condiciones climatológicas adversas y críticas que podrían inducir una desestabilización de la línea de vuelo y el desencadenamiento de situaciones que puedan provocar un accidente.

# 2 OBJETIVOS

## 2.1 OBJETIVO GENERAL

Desarrollar un modelo funcional para el monitoreo cartográfico de las condiciones meteorológicas y las especificaciones técnicas de un vuelo simulado.

## 2.2 OBJETIVOS ESPECÍFICOS

● Mostrar sobre un mapa la última posición geográfica de una aeronave en vuelo.

● Dibujar la trayectoria de una aeronave sobre un mapa.

● Enviar información meteorológica a una aeronave mediante mensajes de texto, como respuesta a solicitudes de reportes meteorológicos.

# 3 MARCO TEÓRICO

## 3.1 CONTEXTUALIZACIÓN DEL PROYECTO

En esta parte del capítulo se presentarán las bases teóricas en las que se fundamenta cada una de las fases del proyecto.

## 3.2 CICLO DE VIDA DEL SOFTWARE

El término ciclo de vida del software se refiere al desarrollo de software, desde la fase inicial hasta la fase final. El propósito de este esquema es definir las distintas fases intermedias que se requieren para validar el desarrollo de la aplicación, es decir, para garantizar que el software cumpla los requisitos para la aplicación y verificación de los procedimientos de desarrollo: se asegura de que los métodos utilizados son apropiados.

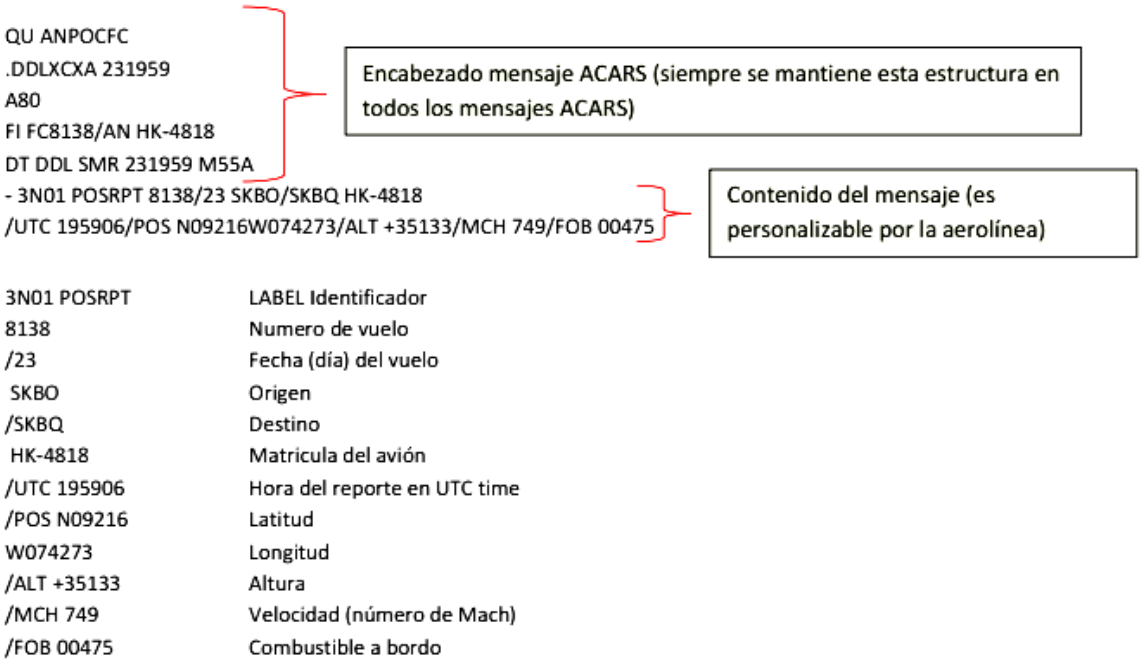
Estos esquemas se originan en el hecho de que es muy costoso rectificar los errores que se detectan tarde dentro de la fase de implementación. El ciclo de vida permite que los errores se detecten lo antes posible y por lo tanto, permite a los desarrolladores concentrarse en la calidad del software, en los plazos de implementación y en los costos asociados.

## 3.3 REPORTES AERONÁUTICOS

En la aviación, haciendo uso de los transpondedores de cada avión, se envían mensajes que contienen las coordenadas (latitud, longitud) y los datos básicos de la posición del avión, a una central que combina tales reportes con lo que sus radares son capaces de identificar, para que de ésta manera se pueda establecer un estimado más exacto de la posición geográfica del avión. Dichos reportes se

transmiten en un formato denominado ACARS[5] cuya estructura se ejemplifica en la siguiente gráfica:

Figura 1. Formato ACARS de transmisión de reporte de posición




Fuente: Autores.

**3.3.1 Mensajes OOOI.** En el reporte ACARS, existe un campo destinado a informar el estado de movilidad en tierra de la aeronave. Como el mensaje es enviado en distintos momentos, podemos interpretar cada reporte como un mensaje distinto, y en ese sentido, estos cuatro mensajes indican los cuatro movimientos básicos que realiza un avión en un aeródromo y que indican que un vuelo ha iniciado y/o está próximo a finalizar, estos mensajes serán el inicio y final del seguimiento de vuelo.

- **OUT REPORT – PUSHBACK**: Indica el inicio del vuelo, el momento en que el avión ya ha cerrado las puertas y empieza a ser remolcado por el tractor.

- **OFF REPORT – TAKEOFF:** Es el momento del despegue, cuando el avión se levanta completamente del suelo.

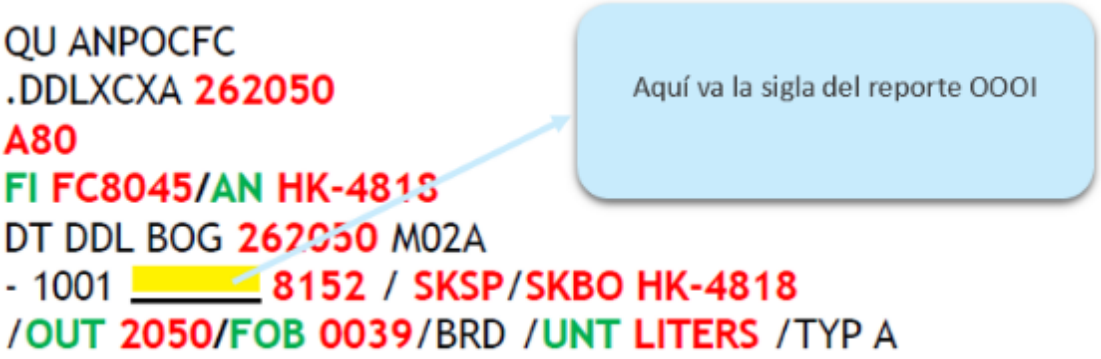
[5] Aircraft Communications Addressing and Reporting System

- **ON REPORT – LANDING**: Indica el aterrizaje, cuando el avión vuelve a tocar la pista en el destino.

- **IN REPORT – ARRIVAL/LANDING:** Es la finalización del vuelo, cuando el avión se detiene en su posición de parqueo.

Cada uno de los anteriores reportes es identificado con las siglas OUTRP, OFFRP, ONRP e INRP respectivamente; los cuales (uno a la vez) se ubican en un segmento del contenido del mensaje.

Figura 2. Formato mensaje OOOI

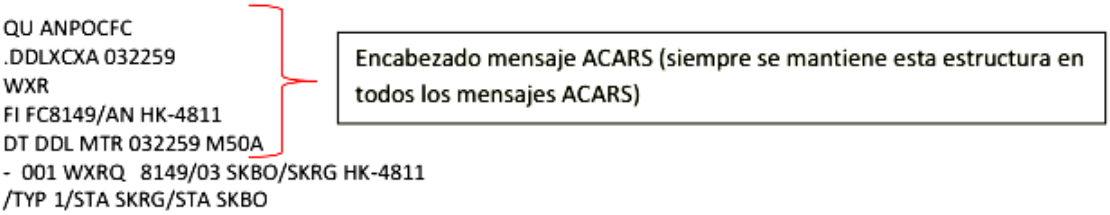



Fuente: Autores

**3.3.2 Solicitud de reporte meteorológico.** METAR es el estándar internacional del formato del código utilizado para emitir periódicamente informes, sobre las observaciones meteorológicas en los aeródromos. Es una sigla traducida del francés (Metéorologique Aviation Régulière) como *Informe meteorológico aeronáutico de rutina* (en inglés: Meteorological Aerodrome Report).

El reporte METAR es usado por los meteorólogos, para ayudarse en los pronósticos del tiempo, y fundamentalmente por los pilotos de las aeronaves para conocer la meteorología de los aeropuertos de destino y actuar en consecuencia. Estos reportes usualmente vienen de los aeropuertos. Típicamente se emiten cada media hora o una hora (depende del aeródromo); sin embargo, si las condiciones cambian significativamente, pueden actualizarse con reportes llamados SPECI.*

Figura 3. Formato ACARS - METAR




| | |
|---|---|
| 001 WXRQ | LABEL Identificador |
| 8149 | Numero de vuelo |
| /03 | Fecha (día) del vuelo |
| SKBO | Origen |
| /SKRG | Destino |
| HK-4811 | Matricula del avión |
| /TYP 1 | Identificador de tipo de reporte solicitado (1=METAR) |
| /STA SKRG | Aeropuerto del que se requiere conocer el METAR (SKRG= Rionegro, Medellín) |
| /STA SKBO | Aeropuerto del que se requiere conocer el METAR (SKBO= Bogotá |

Fuente: Autores.

## 3.4 APLICACIÓN WEB

En la ingeniería de software se denomina aplicación web a aquellas herramientas que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet mediante un navegador. En otras palabras, es una aplicación software que se codifica en un lenguaje soportado por los navegadores web en la que se confía la ejecución al navegador.

Las aplicaciones web son populares debido a lo práctico del navegador web como cliente ligero, a la independencia del sistema operativo, así como a la facilidad para actualizar y mantener aplicaciones web sin distribuir e instalar software a miles de usuarios potenciales

**3.4.1 Entorno CLIENTE/SERVIDOR.** Diversas aplicaciones se ejecutan en un entorno cliente/servidor. Esto significa que los **equipos clientes** (equipos que forman parte de una red) contactan a un **servidor**, un equipo generalmente muy potente en materia de capacidad de entrada/salida, que proporciona servicios a los equipos clientes. Estos servicios son programas que proporcionan datos como la hora, archivos, una conexión, etc.

Los servicios son utilizados por programas denominados **programas clientes** que se ejecutan en equipos clientes. Por eso se utiliza el término "cliente" (cliente FTP, cliente de correo electrónico, etc.) cuando un programa que se ha diseñado para ejecutarse en un equipo cliente, capaz de procesar los datos recibidos de un

servidor (en el caso del cliente FTP se trata de archivos, mientras que para el cliente de correo electrónico se trata de correo electrónico).

**3.4.1.1 Ventajas del entorno CLIENTE/SERVIDOR.** El modelo cliente/servidor se recomienda, en particular, para redes que requieran un alto grado de fiabilidad. Las principales ventajas son:

- **Recursos centralizados**: Debido a que el servidor es el centro de la red, puede administrar los recursos que son comunes a todos los usuarios, por ejemplo: una base de datos centralizada se utilizaría para evitar problemas provocados por datos contradictorios y redundantes.
- **Seguridad mejorada**: Ya que la cantidad de puntos de entrada que permite el acceso a los datos no es importante.
- **Administración al nivel del servidor**: Ya que los clientes no juegan un papel importante en este modelo, requieren menos administración.
- **Red escalable**: Gracias a esta arquitectura, es posible quitar o agregar clientes sin afectar el funcionamiento de la red y sin la necesidad de realizar mayores modificaciones.

**3.4.1.2 Desventajas del entorno CLIENTE/SERVIDOR.** La arquitectura cliente/servidor también tiene las siguientes desventajas:

- **Costo elevado**: Debido a la complejidad técnica del servidor.
- **Un eslabón débil**: El servidor es el único eslabón débil en la red de cliente/servidor, debido a que toda la red está construida en torno a él. Afortunadamente, el servidor es altamente tolerante a los fallos (principalmente gracias al sistema RAID).

**3.4.2 Hypertext Transfer Protocol** o HTTP (protocolo de transferencia de hipertexto) es el protocolo de comunicación que permite las transferencias de información en la World Wide Web[6]. HTTP es un protocolo sin estado, es decir, no guarda ninguna información sobre conexiones anteriores. El desarrollo de aplicaciones web necesita frecuentemente mantener estado. Para esto se usan las cookies, que es información que un servidor puede almacenar en el sistema cliente. Esto le permite a las aplicaciones web instituir la noción de "sesión", y también permite rastrear usuarios ya que las cookies pueden guardarse en el cliente por tiempo indeterminado.

**3.4.3 Mensajes HTTP.** Los mensajes HTTP son en texto plano lo que lo hace más legible y fácil de depurar. Esto tiene el inconveniente de hacer los mensajes más largos.

[6] https://es.wikipedia.org/wiki/Hypertext_Transfer_Protocol (17-11-16)

Los mensajes tienen la siguiente estructura:

- Línea inicial (termina con retorno de carro y un salto de línea) con
    1. Para las peticiones: la acción requerida por el servidor (método de petición) seguido de la URL del recurso y la versión HTTP que soporta el cliente.
    2. Para respuestas: La versión del HTTP usado seguido del código de respuesta (que indica que ha pasado con la petición seguido de la URL del recurso) y de la frase asociada a dicho retorno.
- Las cabeceras del mensaje que terminan con una línea en blanco. Son metadatos. Estas cabeceras le dan gran flexibilidad al protocolo.
- Cuerpo del mensaje. Es opcional. Su presencia depende de la línea anterior del mensaje y del tipo de recurso al que hace referencia la URL. Típicamente tiene los datos que se intercambian cliente y servidor Por ejemplo para una petición podría contener ciertos datos que se quieren enviar al servidor para que los procese. Para una respuesta podría incluir los datos que el cliente ha solicitado.

## 3.5 ARQUITECTURA MVC

El patrón de arquitectura MVC (Modelo Vista Controlador) es un patrón que define la organización independiente del **Modelo** (Objetos de Negocio), la **Vista** (interfaz con el usuario u otro sistema) y el **Controlador** (controlador del workflow de la aplicación).[7]

De esta forma, dividimos el sistema en tres capas donde, tenemos la encapsulación de los datos, la interfaz o vista por otro y por último la lógica interna o controlador.

### 3.5.1 Capas de la arquitectura MVC

#### 1.1.3.1 Modelo

Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). [8]

#### 2.1.3.1 Vista

[7] http://www.lab.inf.uc3m.es/~a0080802/RAI/mvc.html

[8] https://es.wikipedia.org/wiki/Modelo–vista–controlador

Presenta el 'modelo' (información y *lógica de negocio*) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida.

#### 3.1.3.1 Controlador

Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta el 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la vista y el modelo.

## 3.6 HERRAMIENTAS UTILIZADAS EN EL DESARROLLO DEL PROYECTO

A continuación se describen las herramientas utilizadas para el desarrollo de este trabajo de grado.

### 3.6.1 Mapbox

Mapbox es un grupo de utilidades, todas de código libre que sirven para crear mapas. Estas herramientas necesitan que se le ingresen datos para ellasmostrarlos gráficamente. Además, permite utilizar diferentes capas, que pueden superponerse sobre cualquier sector del planeta.

### 3.6.2 Javascript

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas.
Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

### 3.6.3 Json

Formato ligero de intercambio de datos. De fácil lectura y escritura para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript, Standard ECMA-262 3rd Edition - Diciembre 1999. JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++,

C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.

### 3.6.4 Cloud9

Es un ambiente integrado de desarrollo (IDE) que tiene como plataforma la nube. Soporta cientos de lenguajes de programación como C, C++, PHP, Ruby, Perl, Python, JavaScript con Node.js, y Go. Esto permite a los desarrolladores iniciar a codificar inmediatamente con espacios de trabajo pre-configurados, de colaboración con sus compañeros de proyecto.

### 3.6.5 Git

Es un sistema de control de versiones que es usado para el desarrollo de software y otras tareas de control de versiones. Como un sistema distribuido de control de revisión está dirigido a la velocidad, integridad de datos y soporte para flujos de trabajo distribuidos.

### 3.6.6 Mongodb

MongoDB es la base de datos NoSQL líder y permite a las empresas ser más ágiles y escalables. Organizaciones de todos los tamaños están usando MongoDB para crear nuevos tipos de aplicaciones, mejorar la experiencia del cliente, acelerar el tiempo de comercialización y reducir costes. MongoDB ha sido creado para brindar escalabilidad, rendimiento y gran disponibilidad, escalando de una implantación de servidor único a grandes arquitecturas complejas de centros multidatos. MongoDB brinda un elevado rendimiento, tanto para lectura como para escritura, potenciando la computación en memoria. Mongo soporta “rich documents” a diferencia de tablas planas. Permite almacenar “Pre-joint /embebed data” para acelerar tiempos de acceso. No soporta joints, ni constraints, ni definir un esquema rígido.

### 3.6.7 Nodejs

Node es un intérprete Javascript del lado del servidor. Node.js usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. Utiliza un motor JavaScript extremadamente rápido de Google, el motor V8. Utiliza un diseño Orientado por Eventos para mantener el código al mínimo y fácil de leer. Todos estos factores conducen a la meta deseada por Node — es relativamente fácil escribir una solución masivamente escalable.

### 3.6.8 MLab

MLab (mlab.com) es un servicio de base de datos en la nube totalmente administrado que aloja bases de datos MongoDB. MLab se ejecuta en proveedores de Infraestructura Amazon, Google y Microsoft[9].

## 3.7 MARCO METODOLÓGICO

El Proceso para el desarrollo de software, también denominado ciclo de vida del desarrollo de software es una estructura aplicada al desarrollo de un producto de software. Hay varios modelos a seguir para el establecimiento de un proceso para el desarrollo de software, cada uno de los cuales describe un enfoque diferente para diferentes actividades que tienen lugar durante el proceso. Algunos autores consideran un modelo de ciclo de vida un término más general que un determinado proceso para el desarrollo de software.

### 3.7.1 Metodología de desarrollo incremental

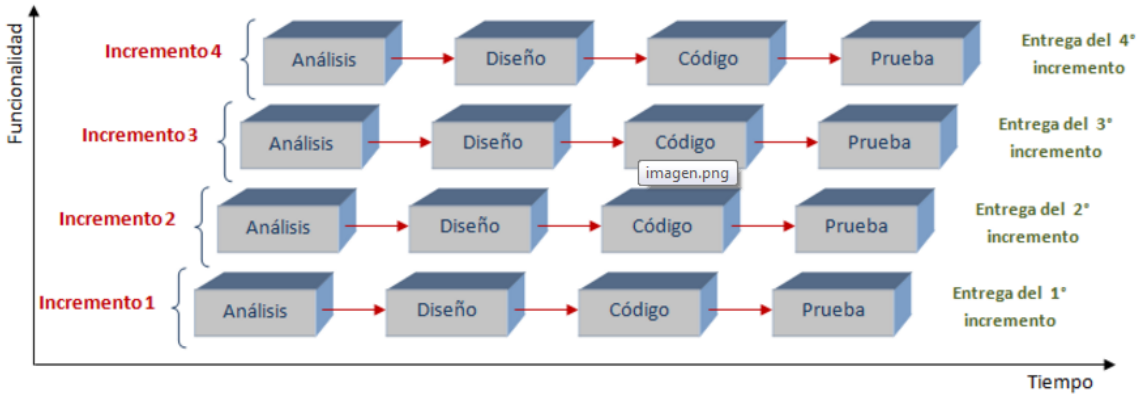
El desarrollo iterativo recomienda la construcción de secciones reducidas de software que irán ganando en tamaño para facilitar así la detección de problemas de importancia antes de que sea demasiado tarde. Los procesos iterativos pueden ayudar a desvelar metas del diseño en el caso de clientes que no saben cómo definir lo que quieren.

Teniendo en cuenta lo anterior y dando la oportunidad de otorgar tiempo a la recolección de requisitos específicos, y bajo la necesidad de obtener productos operacionales en corto tiempo; la metodología de desarrollo incremental, es la que más se ajusta al desarrollo del proyecto. En un desarrollo iterativo e incremental el proyecto se planifica en diversos bloques transitorios llamados iteraciones.

Las iteraciones se conciben como micro-proyectos: en todas las iteraciones se repite un proceso de trabajo similar para facilitar un resultado completo, de manera que el cliente logre alcanzar los beneficios del proyecto de manera incremental. Para esto, es necesario que cada requisito se complete en una única iteración: el equipo debe realizar todas las tareas precisas para consumarlo y que esté apto para ser traspasado al cliente con el minúsculo esfuerzo posible. Así, no quedaría ninguna actividad sensible referida con la entrega de requisitos para la última etapa del proyecto.

[9] Fuente: https://wiki2.org/en/MLab

Figura 4. Esquema del Modelo Incremental




# 4 RESULTADOS DEL PROYECTO

## 4.1 REQUISITOS PARA EL DESARROLLO DE LA HERRAMIENTA

### 4.1.1 requisitos generales

- Mostrar sobre un mapa la última posición geográfica de un avión en vuelo.
- Mostrar sobre un mapa la trayectoria de un avión en vuelo.
- Desplegar información técnica del vuelo (No. Avión, No Vuelo, Lat/Lon, Combustible, Altitud, etc).
- Enviar al avión (simulado) reportes METAR como respuesta a solicitudes enviadas desde la cabina del avión.
- Recibir solicitudes de reportes meteorológicos.
- Recibir reportes ACARS y manipularlos según su propósito.
- Proveer un formulario web que permita enviar mensajes a la plataforma, siendo éste el método que simula el dispositivo emisor de los mensajes desde cada avión a la plataforma.

### 4.1.2 Requisitos no funcionales

- Correr la aplicación sobre un servidor NodeJS.
- Guardar la información en una base de datos MongDB.

### 4.1.3 Requisitos específicos

#### 4.1.3.1 Requisitos primer incremento

- Capturar mensajes de remitentes específicos.

- Seleccionar información relevante de cada mensaje para nutrir el sistema.
- Decodificar el contenido de cada mensaje para ser tratado según su propósito.
- Consultar reportes meteorológicos de una página pública y autorizada.
- Enviar reportes meteorológicos al remitente que los solicite.
- Formatear a JSON el contenido relevante de cada mensaje cuyo propósito sea reportar el despegue, posición o aterrizaje de una aeronave; y disponer una respuesta.

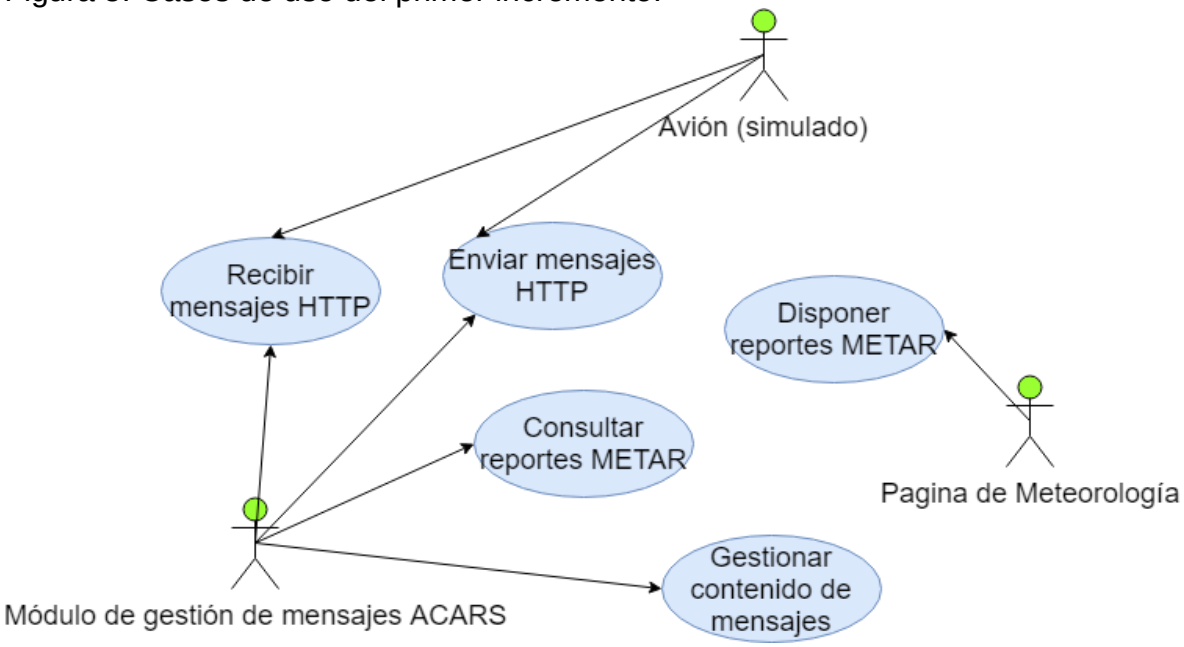
#### 4.3.1.2 requisitos segundo incremento

- Desplegar un mapa con información pertinente a los diferentes aeródromos comerciales de las principales ciudades de Colombia.
- Desplegar sobre el mapa los vuelos que se estén realizando.
- Mostrar información técnica y pertinente a cada vuelo sobre el mapa, al igual que el trazo de su trayectoria.
- Disponer uno o varios servicios para recibir mensajes en formato Json.
- Manipular y abstraer información de los mensajes Json.
- Guardar en Base de Datos cada reporte.
- Mostrar algunas gráficas o tablas pertinentes a los vuelos.
- Disponer información meteorológica actualizada de los principales aeropuertos de Colombia.

## 4.2 CASOS DE USO

**4.2.1 Casos de uso primer incremento.** A continuación se visualizará cada uno de los casos de uso implementados para el desarrollo del sistema en su primera etapa.

Figura 5. Casos de uso del primer incremento.




Fuente. Autores.

#### 4.2.1.1 Descripción de casos de uso

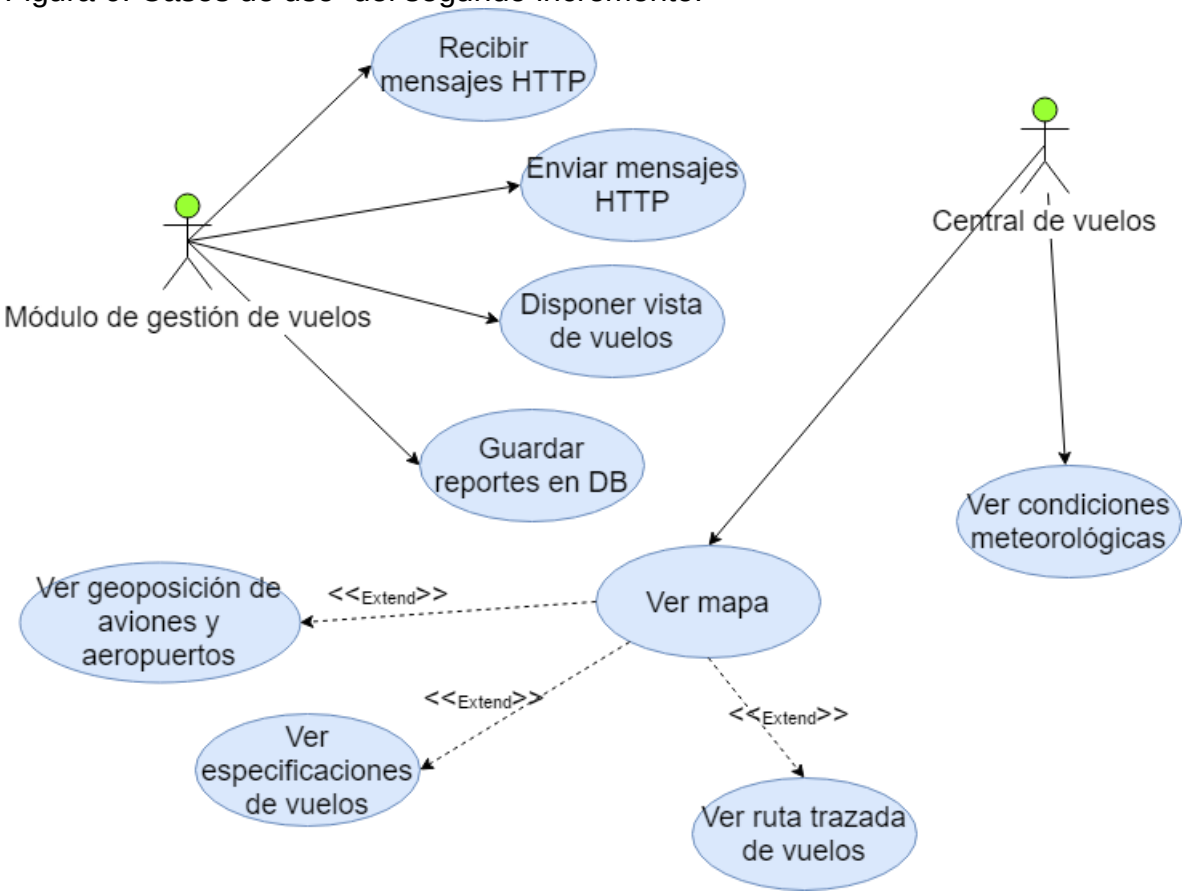
| **Caso de uso: Recibir/Enviar mensajes HTTP, Gestionar contenido de mensajes** | |
|---|---|
| **Actor: Avión** | **Actor: Sistema** |
| • El avión (simulado) envía un mensaje | • El sistema recibe el mensaje<br>• Devuelve un Status 200<br>• Procesa el contenido del mensaje ACARS<br>• Dispone el contenido del mensaje como un servicio en formato JSON |
| **Flujos alternos** | |
| • El mensaje puede no ser válido, es decir, puede no tener la estructura ACARS correcta. En ese caso el sistema devuelve un mensaje indicando este suceso y la información no es procesada. | |

| **Caso de uso: Consultar/Disponer reportes METAR** |
|---|

| Actor: Sistema | Actor: Página de reportes meteorológicos |
|---|---|
| • **Se recibe una solicitude de reporte METAR**<br>• Mediante un algoritmo tipo Web Crawler el sistema consulta la página que contiene los reportes meteorológicos en formato METAR.<br>• El sistema envía el reporte METAR como un mensaje HTTP. | • La página autorizada dispone del reporte. |
| **Flujos alternos** | |
| • La ip o url de la página puede cambiar. | |

### 4.2.2 Casos de uso segundo incremento

Figura 6. Casos de uso  del segundo incremento.




Fuente. Autores.

#### 4.2.2.1 Descripción de casos de uso

| **Caso de uso: Ver mapa, Ver geoposición de aviones y aeropuertos** | |
|---|---|
| **Actor: Central de Vuelos** | **Actor: Sistema** |
| • El usuario encargado abre su navegador y teclea el dominio asignado para el sistema. | • El sistema despliega en su página principal un mapa con los íconos de los aviones y aeropuertos (según su posición) cubiertos por la aerolínea. |
| **Flujos alternos** | |
| • El navegador podría ser obsoleto, por tanto los gráficos podrían presentar problemas. | |

| Caso de uso: Ver ruta trazada de vuelos | |
|---|---|
| **Actor: Central de Vuelos** | **Actor: Sistema** |
| • El usuario se ubica (sin pulsar) sobre el ícono de un avión. | • Se detecta el evento (de Javascript) y despliega la ruta del vuelo a partir de los reportes, junto con una recta origen-destino. |
| **Flujos alternos** | |
| | |

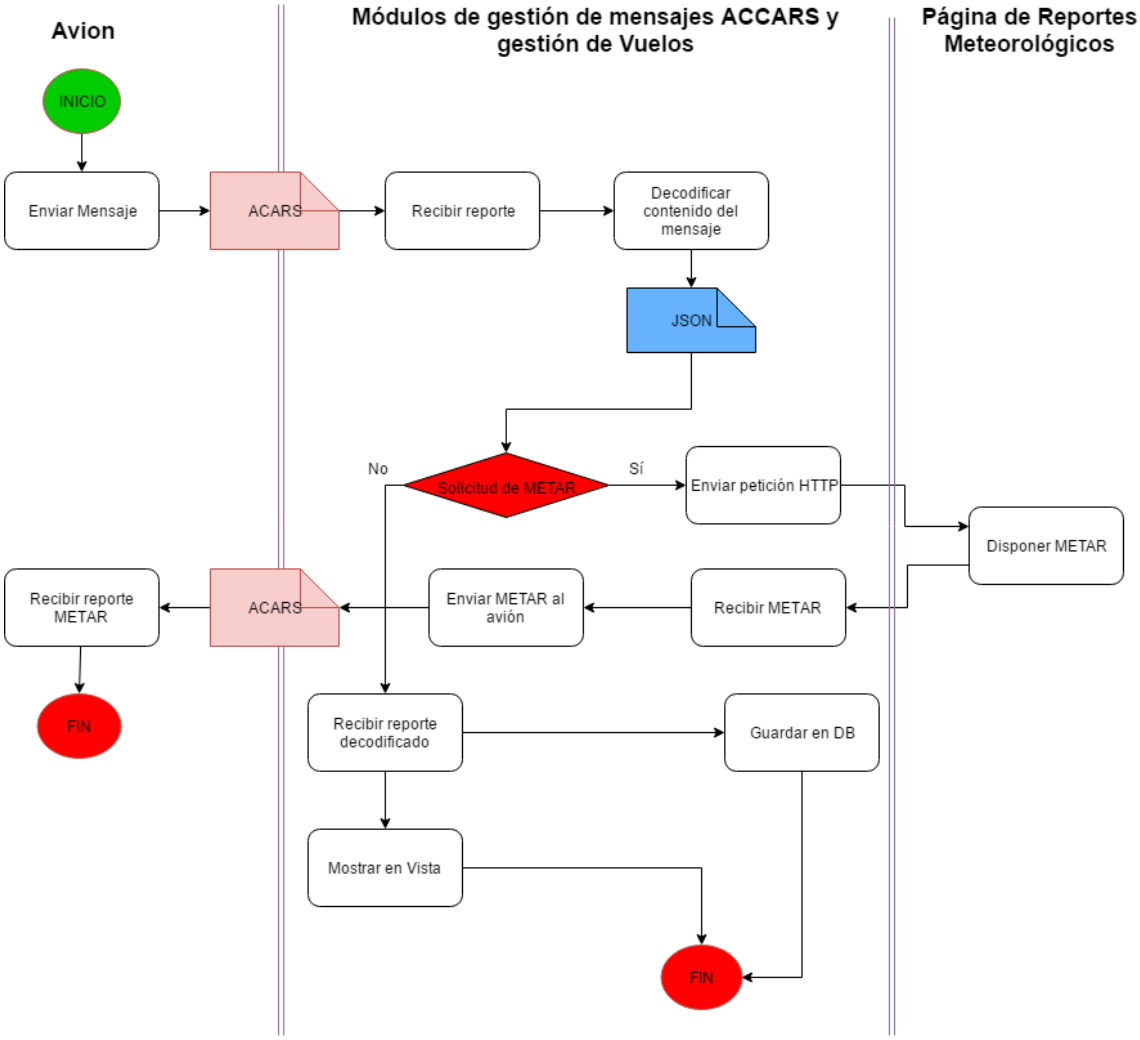
| Caso de uso: Ver especificaciones de vuelo | |
|---|---|
| **Actor: Central de Vuelos** | **Actor: Sistema** |
| • Hacer click sobre el ícono de un avión.<br>• Pulsar el botón que señala el vuelo. | • Se muestra un cuadro sobre el ícono del avión que muestra el número de vuelo dentro de un botón.<br>• Aparece un panel en la parte izquierda de la pantalla con la información técnica del vuelo. |
| **Flujos alternos** | |
| • El navegador podría ser obsoleto, por tanto los gráficos podrían presentar problemas. | |

### 4.3 DIAGRAMA DE ACTIVIDADES

Los diagramas de actividades sirven para representar el comportamiento dinámico de un sistema haciendo hincapié en la secuencia de actividades que se llevan a cabo y las condiciones que guardan o disparan esas actividades.

El siguiente diagrama representa el flujo normal de las actividades principales de nuestro sistema, exponiendo los formatos o tecnologías en que son transportados los datos más importantes.

Figura 7. Diagrama de Actividades.

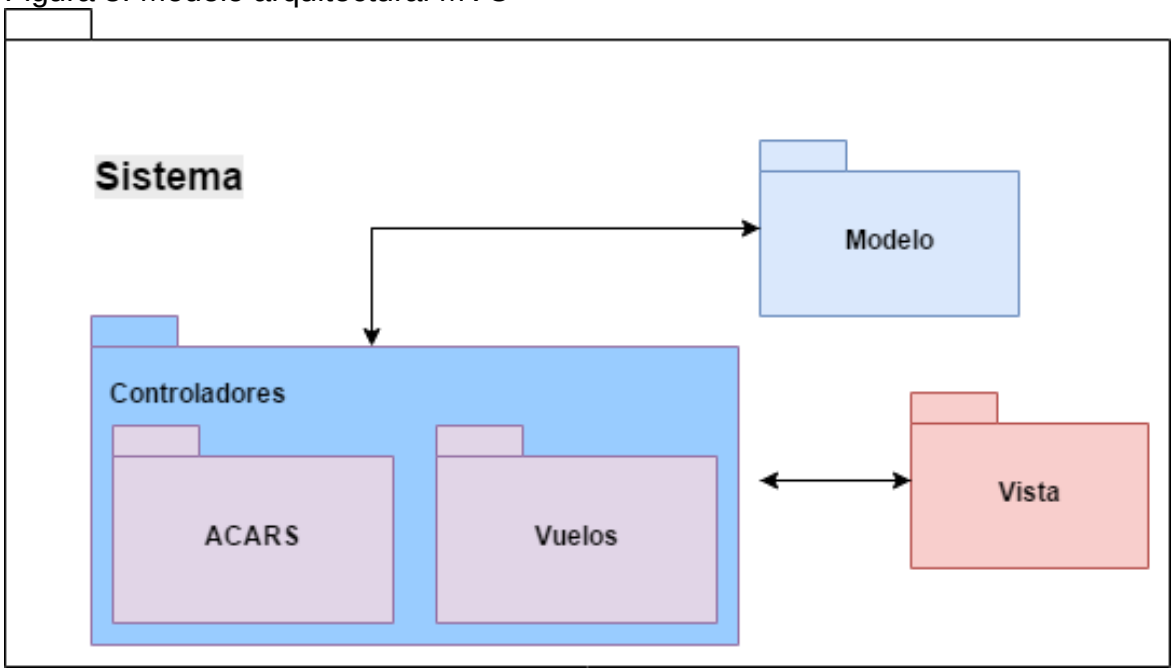



Fuente: Autores.

## 4.4 ARQUITECTURA DEL SISTEMA

Hemos separado el sistema en dos módulos básicamente por cuestiones de escalabilidad y flexibilidad. Puesto que los estándares del formato ACCARS pueden cambiar en cualquier momento, o quizá se quiera implementar el manejo de la misma información bajo otros estándares, o si es necesario aumentar en funcionalidades, pensamos que es conveniente dividir la lógica, y servirse de un módulo como un controlador intermedio.

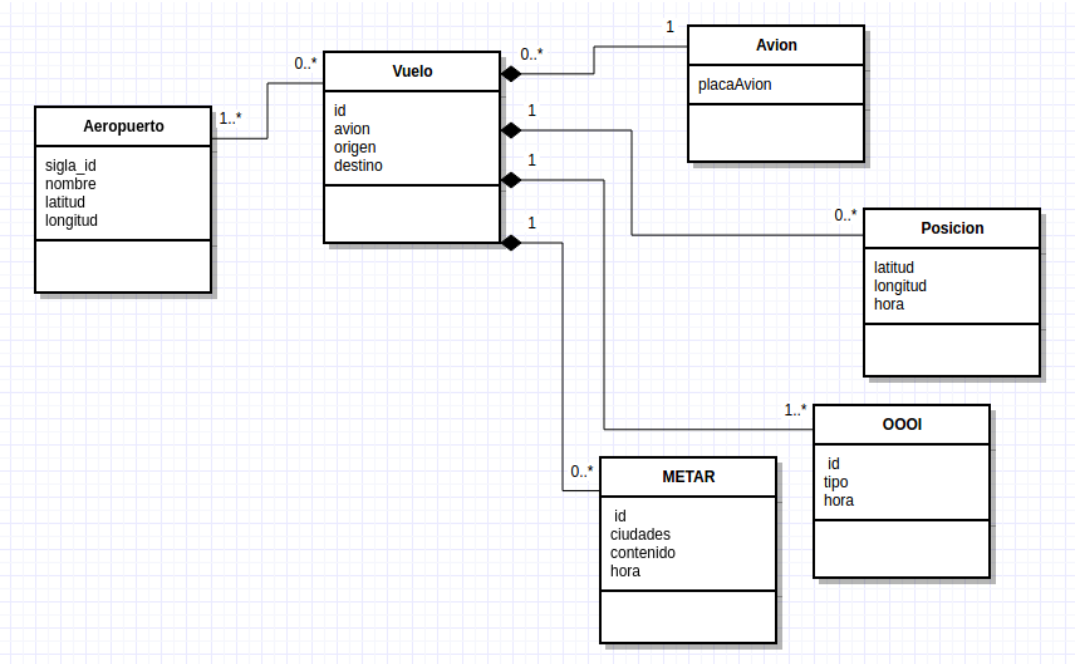
Figura 8: Modelo arquitectural MVC




Fuente. Autores.

## 4.5 DIAGRAMA DE CLASES

Figura 9. Diagrama de clases.




Fuente. Autores.

## 4.6 DESARROLLO DE LA APLICACIÓN WEB

### 4.6.1 Entorno de Desarrollo Integrado (IDE)

Un entorno de desarrollo integrado o entorno de desarrollo interactivo, en inglés Integrated Development Environment (IDE), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software. Normalmente, un IDE consiste de un editor de código fuente, herramientas de construcción automáticas y un depurador[10].

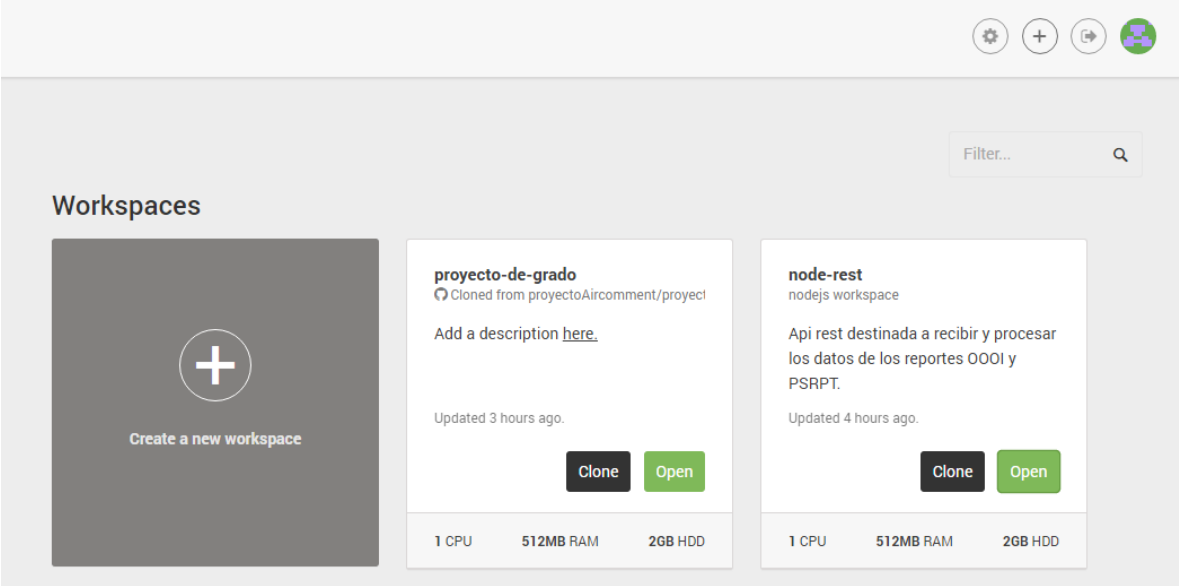
El desarrollo del sistema se llevó a cabo sobre Cloud9, un servicio de **plataforma en la nube (PAAS),** que no sólo consta de un IDE de desarrollo, sino también de un contenedor sobre el cual depurar, correr y probar a cabalidad el sistema.

Cloud9 dispone una máquina virtual pública con Sistema Operativo Linux (Ubuntu) para cada proyecto. Así desde el navegador y a través de internet puede accederse a la terminal como usuario ROOT, con todos los permisos y casi todos los programas de consola que podemos encontrar en un sistema Linux instalado en

[10] https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado

una máquina local, y además con muchas librerías de desarrollo para empezar a trabajar en la aplicación desde el principio.

Figura 10. Interfaz de visualización de proyectos en Cloud9.




Fuente. Tomado de cloud9.com.

En nuestro caso, además de almacenar y correr nuestros dos módulos sobre la plataforma de cloud9, hemos creado un repositorio en GitHub, para un manejo eficiente versiones, y al cual se puede acceder y gestionar desde la terminal provista por c9.

Sobre éste contenedor se desarrolló un sistema en lenguaje Javascript sobre un servidor NodeJS.

**4.6.2 Primer incremento.** Se desarrolló un sistema tipo REST que recibe mensajes en formato ACARS y los desglosa para deisponerlos como JSON.

**4.6.3 Segundo incremento.** Se encarga de comunicarse con el modelo para guardar la información en la base de datos y de generar las vistas respectivas.

**4.6.4 Base de datos.** Para este proyecto se optó para la gestión de los datos un DBMS noSQL llamado MongoDB. MongoDB es una base de datos ágil que permite a los esquemas cambiar rápidamente cuando la aplicación evoluciona, proporcionando siempre la funcionalidad que esperamos de las bases de datos tradicionales, tales como índices secundarios, un lenguaje completo de búsquedas y consistencia estricta, como también la tolerancia a fallos automática, que nos ofrece fiabilidad y flexibilidad operativa, que para este caso puntual es el factor

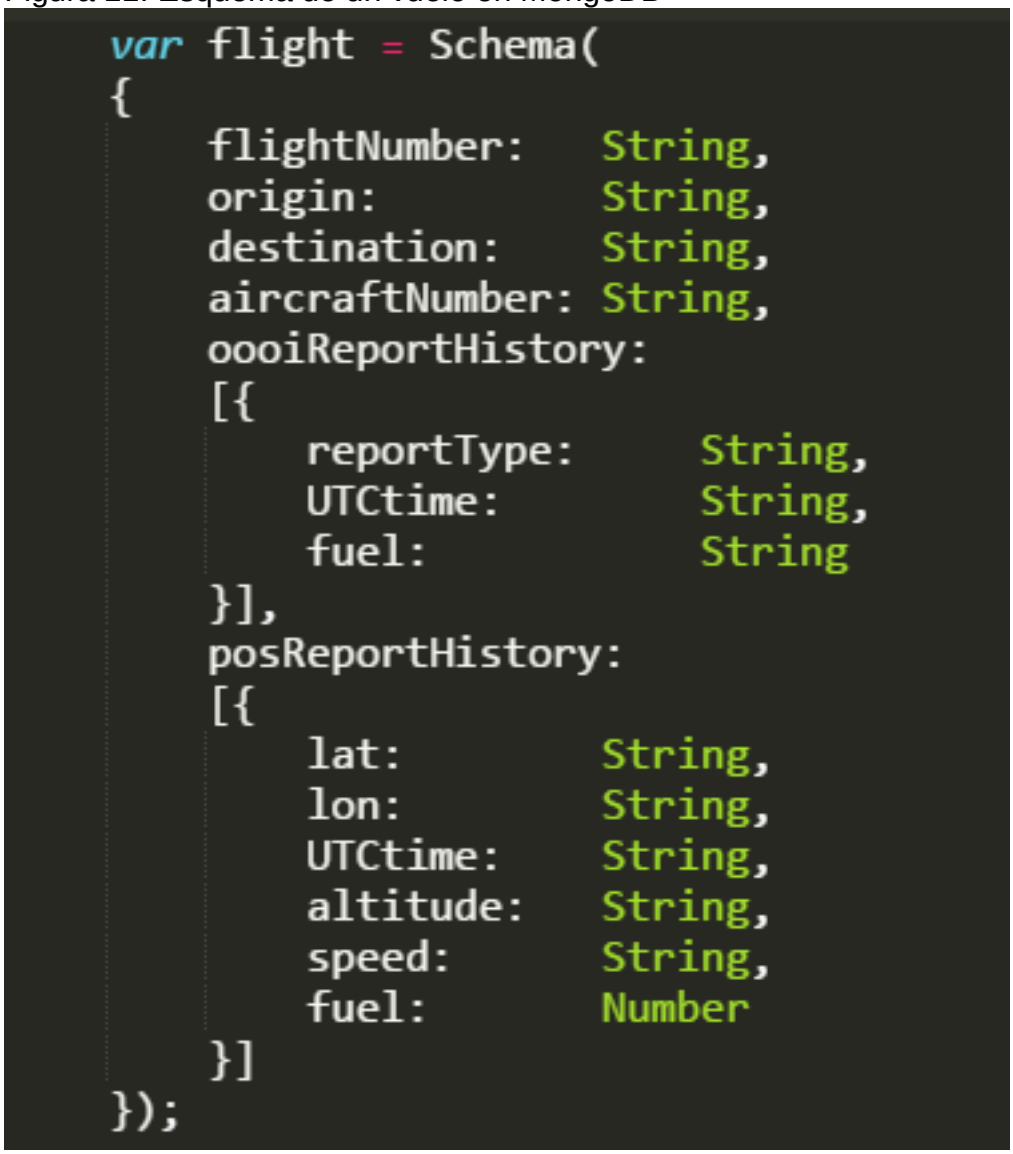
trascendental[11].

Teniendo en cuenta el alcance de este proyecto, hacer uso ‘total’ del modelo relacional para que gestione las posiciones de cada uno de los distintos vuelos en cuestión, haría que las tablas crecieran de forma desmesurada, ralentizando las peticiones de los recursos y por ende entorpeciendo el foco de este proyecto, que es el despliegue de información en tiempo real. Por ello, y teniendo en cuenta estos factores, éste proyecto se decanta por gestionar tal información en documentos, cuya arquitectura es gestionada a través de esquemas desde la capa ‘Modelo’, permitiendo escalabilidad a futuro, pues los esquemas pueden cambiar si el sistema lo requiere, y MongoDB no se opone a tales cambios estructurales, brindando rendimiento y gran disponibilidad al crecimiento.

Para cada vuelo corresponde un único documento, donde almacena de forma dinámica todos los datos pertinentes a dicho vuelo, pudiendo realizar operaciones atómicas sobre el documento.

[11] Fuente: https://www.mongodb.com/es

Figura 11. Esquema de un vuelo en MongoDB



```
var flight = Schema(
{
    flightNumber:   String,
    origin:         String,
    destination:    String,
    aircraftNumber: String,
    oooiReportHistory:
    [{
        reportType:     String,
        UTCtime:        String,
        fuel:           String
    }],
    posReportHistory:
    [{
        lat:        String,
        lon:        String,
        UTCtime:    String,
        altitude:   String,
        speed:      String,
        fuel:       Number
    }]
});
```

Fuente**:** Autores

Como podemos apreciar la estructura en la figura (10) se representan las clases: Vuelo (flight), OOOI (oooiReportHistory) y Posición (posReportHistory). De esta forma un vuelo contiene una secuencia de posiciones y reportes OOOI guardados en vectores dentro de su propio cuerpo. Origen y destino, hacen referencia a un documento de Aeropuertos, en los que se encuentran las especificaciones de cada uno.

Cabe destacar que los reportes METAR no son almacenados en ningún caso, puesto que sólo son útiles para el piloto que los solicita.

A continuación se muestra un ejemplo de un documento obtenido como respuesta a una inserción de un nuevo reporte de posición correspondiente a un vuelo:

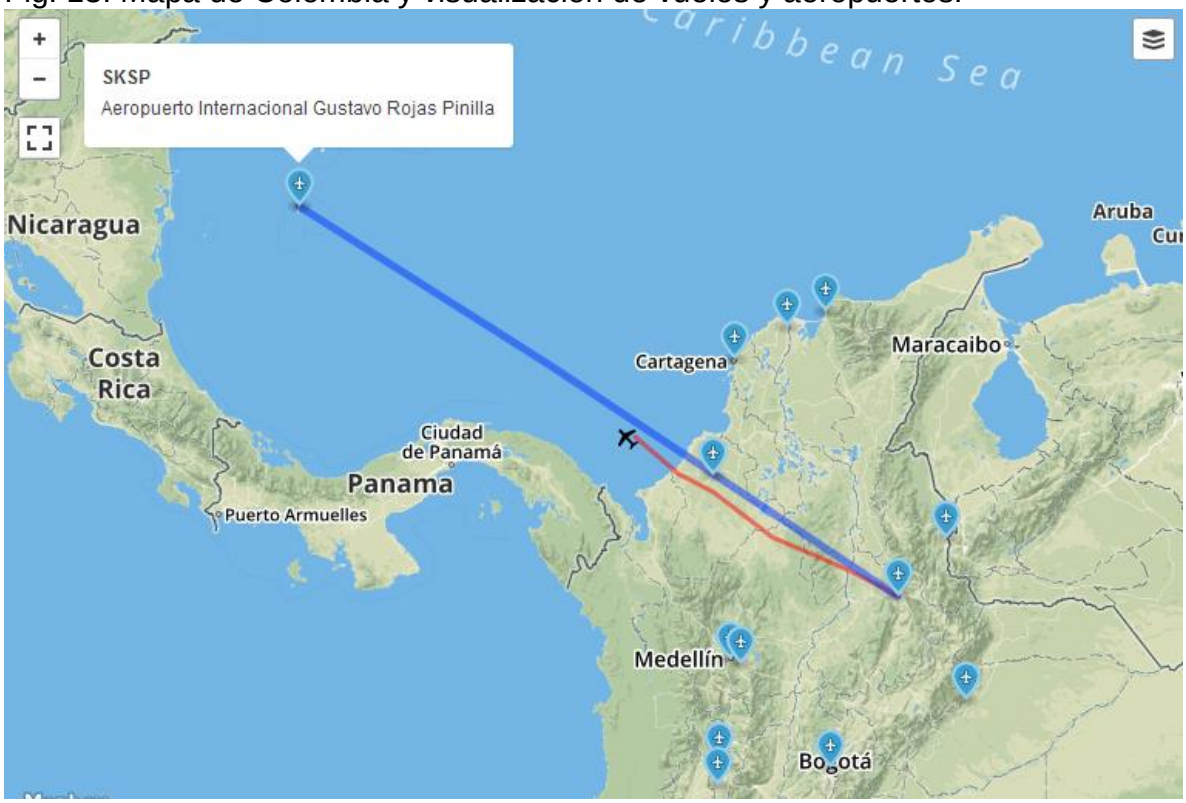
Figura 12. Respuesta de un reporte añadido exitosamente.


```
{
   message: "El reporte ha sido añadido correctamente.",
 - model: {
      _id: "5878b656053e0c0b376c3bbb",
      aircraftNumber: "HK-4818",
      destination: "SKBO",
      origin: "SKSP",
      flightNumber: "8000",
      __v: 0,
   - posReportHistory: (3)[
      - {
            fuel: 475,
            speed: "749",
            altitude: "+35133",
            UTCtime: "195906",
            lon: "-74.455",
            lat: "9.36",
            _id: "5878b73e053e0c0b376c3bbe"
         },
      + {...},
      + {...}
      ],
   - oooiReportHistory: (4)[
      - {
            fuel: "0039",
            UTCtime: "2050",
            reportType: "OUTRP",
            _id: "5878b656053e0c0b376c3bbc"
         },
      + {...},
      + {...},
      + {...}
      ]
   }
}
```


El campo ‘model’ hace referencia al modelo de datos de un vuelo y su contenido, y el campo ‘message’ indica que el reporte fue añadido con éxito al documento de MongoDB.

**4.6.5 Mapa.** A continuación se muestra la evolución en el desarrollo de la vista principal para el usuario operador del sistema.

Fig. 13. Mapa de Colombia y visualización de vuelos y aeropuertos.




## 4.7 PRUEBAS E IMPLEMENTACIÓN

Para la etapa de pruebas e implementación, con la unión de todos los componentes de cada una de las fases de desarrollo para un sistema de información, se colocó en funcionamiento el sistema.

Las pruebas funcionales se enfocan en comprobar que los sistemas desarrollados funcionan acorde a las especificaciones funcionales y requisitos del cliente. Este proceso ayuda a los proyectos de ingeniería de software a detectar los posibles defectos derivados de errores en la fase de programación.

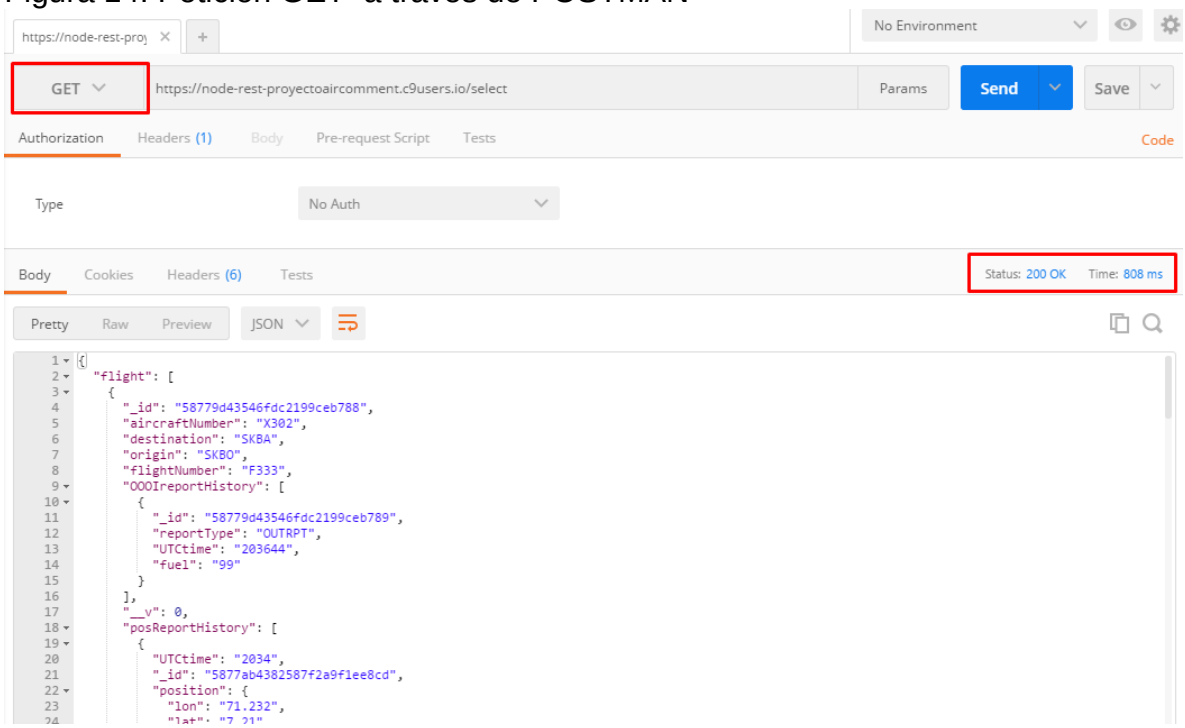
Para la estabilidad del proyecto, se hace galante emplear diferentes tipos de pruebas para distintos objetivos:

- Asegurar que la funcionalidad entregada se amolde a los requisitos del cliente.

- Identificar errores, es decir, se cumple la funcionalidad pero no puede concluir por un error.
- Identificar escenarios de ejecución no previstos por el usuario.
- Asegurar que la aplicación presta servicio bajo parámetros aceptables de desempeño (tiempo de respuesta), que soporta la carga y no colapsa.
- Asegurar que el código desarrollado es confiable, mantenible y puede crecer con la aplicación.

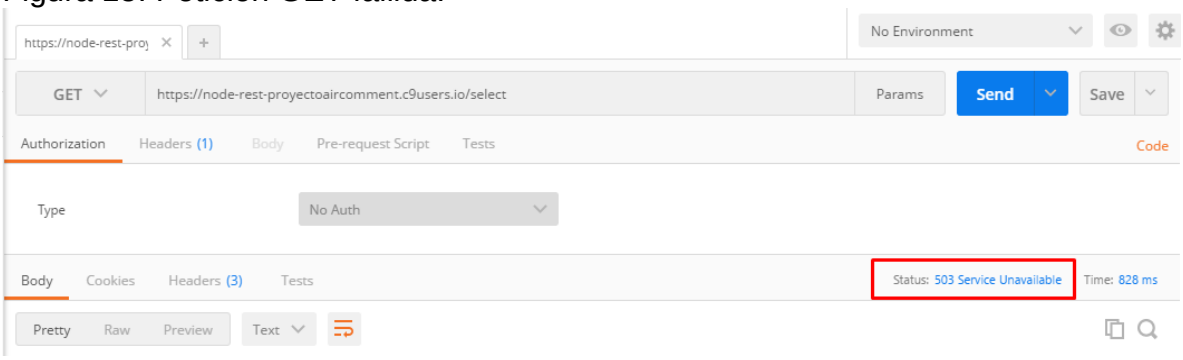
**4.7.1 Pruebas HTTP con Postman.** Postman está capacitado para hacer peticiones GET, POST, PUT y DELETE. Uno de los servicios que nuestra API REST entrega, es el de la selección del historial de vuelos del día. A continuación se muestra el resultado de ejecutar la petición mediante Postman.

Figura 14. Petición GET a través de POSTMAN




Esta misma petición nos entregó el siguiente resultado al presentarse un error en el servidor.

Figura 15. Petición GET fallida.




Previo a la integración de los dos módulos fue necesario asegurarnos de que estas pruebas fueran efectivas para todos los métodos http.

Luego se hicieron pruebas con todo el sistema integrado, incluyendo el envío real de mensajes de texto.

# 5 CONCLUSIONES

El modelo funcional creado, intenta proveer un método de solución óptimo y económico para algunas compañías de aviación que aún no automatizan ciertos protocolos de comunicación. Nuestro modelo, que es apenas una parte del conglomerado necesario para un sistema completo de comunicaciones avión-central, involucra varias tecnologías y presenta varios escenarios en los que se pueden presentar errores, básicamente por interrupciones de comunicación o caídas de servidor.

El encargado de recibir Mensajes de texto es una plataforma externa al sistema, por tanto nuestra aplicación únicamente se encarga de recibir los reportes de los mensajes enviados mediante el protocolo HTTP.
Para una implementación real es necesario crear un sistema de recuperación de fallos robusto y valerse de una infraestructura óptima, pues la información que se maneja es muy importante. Básicamente resumimos que para la arquitectura implementada (cliente/servidor) el mayor riesgo de fallos está en la caída de alguno de los servidores.

## BIBLIOGRAFIA


Huerta, J. M. (Junio de 2016). *ECMAScript.* Obtenido de ECMA Internacional Org, Web: http://academic.uprm.edu/jhuerta/HTMLobj-223/Estudio_de_Necesidades.pdf

ECMA Internacional (Junio de 2016). *ECMAScript.* Obtenido de ECMA Internacional Org: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

IEEE Computer Society. (20 de Octubre de 1998). IEEE Std 830-1998. *IEEE Recommended Practice for Software Requirements Specifications*.

Jeltsch F., E. R. (11 de Noviembre de 2011). *El ciclo de desarrollo de los sistemas de información.* Obtenido de Dr. Eric R. Jeltsch F Web Site: http://dns.uls.cl/~ej/web_Si_2011/Lect_Si_2011/desarrollo-Siste.docx

Kendall, K. E., & Kendall, J. E. (2005). *Análisis y diseño de sistemas* (Sexta ed.). México: Pearson Educación.

McConnell, Steve. «7: Lifecycle Planning». *Rapid Development*. Redmond, Washington: Microsoft Press. p. 140. *Metodología para el desarrollo de estudios organizacionales.* (s.f.). Recuperado el 9 de Noviembre de 2016, de Red de bibliotecas Landivarianas: http://biblio3.url.edu.gt/Libros/org_empresas/1.pdf

Pressman, R. ( Marzo 4 2010). *Software Engineering A Practitioner's Approach. 7ma Edicion, Capitulo 2.3.3 EvolutionaryProcessModels* (Septima edición ed.). España: McGraw-Hill Interamericana Editores S.A. de C.V.

Tentor, J. (11 de Diciembre de 2008). *Arquitectura de N-Capas y N-Niveles.* Obtenido de Software y Aplicaciones Web: Blog de desarrollo de software y aplicaciones web: http://www.jtentor.com.ar