Universidad de Ciencias Informáticas

Facultad 1 - Centro de Identificación y

Seguridad Digital CISED

Trabajo de Diploma en opción al título de Ingeniero en Ciencias Informáticas

Sistema de Gestión de Licencias del Personal Aeronáutico del Instituto de Aeronáutica Civil de Cuba.

Autor: Alejandro Fernández Feitó

Tutores: MSc. Dismey Saavedra López

Ing. Rosana Castro García

Junio 2017

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____
Alejandro Fernández Feitó

Firma del autor

_____ _____
MSc. Dismey Saavedra López Ing. Rosana Castro García

Firma del tutor Firma del tutor

Tenemos que hacer una universidad de excelencia. Debemos tener para esta Universidad una tropa élite de la juventud, para que trabaje allí (...)"

Fidel Castro Ruz. 15 de agosto de 2002.

Dedicatoria

A mi madre:

por todos estos años de sacrificio y dedicación para que yo haya podido tener y ser todo lo que hasta hoy he logrado.

A mi padre:

que a pesar de la distancia siempre lo tengo muy cerca y me motiva prepararme y crecer tanto en conocimiento como en persona.

Agradecimientos

A mis padres, por su confianza y apoyo incondicional que siempre me han brindado para lograr mis metas y cumplir mis sueños.

A mis tutoras Dismey y Rosana, que siempre me han ayudado mucho y toda la paciencia que han tenido conmigo, dos excelentes personas las cuales considero las mejores tutoras que a alguien le puede tocar.

A Gregorio por compartir sus conocimientos.
A mis amigos de la UCI que siempre llevaré en el corazón.

Resumen

El Instituto de Aeronáutica Civil de Cuba es el organismo encargado de dirigir y controlar la política del Estado cubano en cuanto a transporte aéreo. Este instituto para la gestión de licencias de pilotos y habilitaciones actualmente utiliza un sistema que no cubre las necesidades demandadas, por lo que debido a la importancia que han tenido estas aplicaciones en la eficiencia de la gestión empresarial, se decidió crear un nuevo sistema en ambiente *web,* plataforma que logra obtener mayor beneficio dadas las necesidades del cliente. La solución propuesta consiste en desarrollar un sistema informático que permita llevar a cabo la gestión y emisión de los documentos de licencias del personal que se realiza en la Oficina de Licencias del Instituto de Aeronáutica Civil de Cuba. Para ello se realizó una investigación de un grupo de sistemas informáticos similares y funcionales que actualmente se utilizan en el ámbito nacional e internacional; se seleccionó una metodología de desarrollo para regir la organización de la investigación; se seleccionaron las diferentes herramientas para llevar a cabo la propuesta de solución y fueron implementados los requisitos funcionales logrando un sistema que cumple con todas las especificaciones del usuario. Su realización posibilita un mejoramiento en las condiciones de trabajo del cliente y la eficacia de los procesos realizados en la oficina de licencias, permitiendo la obtención de resultados con la calidad requerida actualmente.

Palabras clave: habilitación, licencia aeronáutica, personal aeronáutico, restricción.

Índice

Introducción 1
Capítulo 1. Fundamentación teórica. 5
1.1 Introducción 5
1.2 Conceptos fundamentales 5
1.3 Sistemas de gestión de licencias 6
1.4 Sistemas Homólogos 6
1.4.1 A nivel internacional 6
Sistema de Licencias Aeronáuticas de Chile (ALVI) 7
Sistema Informático del Personal Aeronáutico(SIPA) de Ecuador 7
1.4.2 A nivel nacional 8
Sistema para el control de licencias y habilitaciones 8
1.5 Metodologías de Desarrollo de *Software* 9
Metodologías Ágiles 9
Metodologías Tradicionales 11
1.6 Servicios *web* 12
1.7 Análisis de las tecnologías, herramientas y lenguajes a utilizar 12
1.7.1 Lenguajes 12
1.7.2 Herramientas 14
1.7.3 Gestores de Base de datos 15
1.7.4 Framework 17
1.7.5 Entorno de desarrollo integrado(IDE) 18
1.7.6 Herramientas Case 19
1.7.7 Selección de las tecnologías 19
1.8 Conclusiones parciales 20
Capítulo 2. Análisis y diseño de la propuesta de solución. 21
2.1 Introducción 21
2.2 Modelo del dominio 21
2.3 Propuesta de Solución 22
2.4 Especificación de los Requisitos del *Software* 23
2.4.1 Requisitos funcionales 24
2.4.2 Requisitos no funcionales 26
2.5 Historias de usuario 27
2.6 Estilo Arquitectónico 30

2.7 Arquitectura 31
2.8 Diagrama de clases del diseño 32
2.9 Patrones de diseño 32
2.10 Modelo de Datos 35
2.11 Modelo de despliegue 37
2.12 Conclusiones parciales 37
Capítulo 3. Implementación y prueba. 39
3.1 Introducción 39
3.2 Modelo de componentes que integran la solución informática 39
3.3 Diagrama de componentes 39
3.4 Estándar de codificación 40
Convenciones 41
3.4.1 Nomenclatura según el tipo de clases 41
3.4.2 Nomenclatura de las funcionalidades y atributos 42
3.5 Interfaces de Usuario 42
3.6 Pruebas y resultados 43
3.6.1 Pruebas unitarias 44
3.6.2 Pruebas funcionales 45
3.6.3 Pruebas de Seguridad 48
3.7 Conclusiones parciales 51
Conclusiones 52
Bibliografía 53

Índice de Imágenes

Imagen 1 Diagrama Modelo del dominio 22
Imagen 2 Propuesta de Solución 23
Imagen 3 Historia de Usuario(Adicionar habilitación) 29
Imagen 4 Arquitectura MVC 31
Imagen 5 Diagrama de clases del diseño del RF adicionar habilitación 32
Imagen 6 Patrón de diseño Experto de la clase TipoLicenciaController 34
Ilustración 7 Patrón de diseño controlador de la clase HabilitacionController 35
Imagen 8 Modelo de datos 36
Imagen 9 Modelo de Despliegue 37
Imagen 10 Diagrama de componentes 40
Imagen 11 Login 43
Imagen 12 Interfaz de usuario 43
Imagen 13 Resultados de prueba unitaria sobre MbroTripulacionController Iteración 1 44
Imagen 14 Resultados de prueba unitaria sobre NomencladorController Iteración 1 .. 45
Imagen 15 Resultados de prueba unitaria sobre NomencladorController Iteración 2 .. 45
Imagen 16 Prueba de seguridad iteración 1 (Acunetix) 50
Imagen 17 Prueba de seguridad iteración 2 (Acunetix) 50
Imagen 18 Prueba de seguridad iteración 3 (Acunetix) 51

Índice de Tablas

Tabla 1 Diferencia entre tecnologías ágiles y tradicionales. Fuente, (Avante, 2013) ... 11
Tabla 2 Conceptos del modelo del dominio ... 22
Tabla 4 Variables del caso de prueba para el escenario Adicionar Tipo de Licencia ... 46
Tabla 5 Casos de prueba Introducir Tipo de Licencia ... 47

Introducción

El desarrollo vertiginoso de las Tecnologías de la Información y las Comunicaciones (TIC) como instrumentos de desarrollo, el auge del comercio electrónico, las formas de organización que ha adquirido el ser humano y el modo en que se desenvuelve en la sociedad del presente siglo, demandan la automatización y personalización de procesos de disímiles empresas, para elevar la calidad de los servicios, tornarlos más ágiles, de menor costo y fortalecer el control. Estos argumentos son cimientos que han sostenido el constante desarrollo en el ámbito tecnológico, principalmente en el área del *software*, desarrollándose estos con una nueva óptica y haciendo uso de nuevas arquitecturas. Entre estas se destaca el uso de las arquitecturas *web*, con estándares y protocolos que viabilizan la interoperabilidad de las aplicaciones sobre la red. (Feitó Cabrera, y otros, 2012)

Cuba es un país que se mantiene actualizado del progresivo avance tecnológico a nivel internacional y aún sin ser desarrollado, cuenta con las tecnologías y el personal calificado para introducir progresivamente el uso de las TIC en los sectores de la sociedad, con el objetivo de ganar eficiencia y eficacia en la gestión de los procesos. Una institución cubana que siempre ha estado al tanto de los avances tecnológicos es la Universidad de las Ciencias Informáticas (UCI), que como principal precursora de la idea del Comandante en Jefe Fidel Castro de informatizar la sociedad cubana, despliega muchos de los sistemas implantados en ella en diversas instituciones del país. (Feitó Cabrera, y otros, 2012)

Entre los principales objetivos de la UCI se encuentra la producción de *software*, ya que, en la misma, en su modelo de formación, incluye la vinculación estudio-trabajo. Para esto se apoya en diferentes centros creados por la Universidad, entre los que se encuentra el Centro de Identificación y Seguridad Digital (CISED) de la Facultad 1.Entre los proyectos en desarrollo que tiene CISED se encuentra el Sistema de Gestión de Licencias del Personal Aeronáutico del Instituto de Aeronáutica Civil de Cuba, único en su tipo en el país, que se encarga de emitir las licencias de los titulares autorizados en dependencia de las habilitaciones (clasificaciones) que poseen.

El Instituto de Aeronáutica Civil de Cuba (IACC) es el organismo encargado de dirigir, ejecutar y controlar la política del Estado en cuanto a transporte aéreo, la navegación aérea y sus servicios auxiliares y conexos; para ello dirige y promueve el servicio profesional y especializado en todos los niveles de la organización, con el objetivo de apoyar el crecimiento sostenido del turismo, manteniendo el comportamiento ético que

rige la aviación internacional, caracterizado por una alta fiabilidad, rapidez, seguridad y eficiencia, sustentados en la lealtad, compromiso y comunicación abierta de la dirección y los empleados con los clientes. (IACC, 2017)

Actualmente el sistema implantado en el IACC no cumple con todas las necesidades del cliente, al carecer de funcionalidades que le impiden gestionar con mayor eficacia el proceso, como son, no permite la gestión del personal del instituto (solo permite la gestión de pilotos); las licencias emitidas no cumplen con los estándares a nivel mundial establecidos por la Organización Aeronáutica Civil Internacional (OACI) en cuanto al formato y la traducción de los parámetros; está montado sobre herramientas obsoletas, está programado en Delphi y utiliza como gestor de base de datos la herramienta Microsoft Access 2003, la dependencia de herramientas tan antiguas hace que, por problemas de compatibilidad con los actuales *Sistemas Operativos* de *Windows* sea obligatorio utilizar *Windows XP*, siendo este, una versión antigua a la cual ya no se le brinda soporte, presentando así graves problemas de seguridad para la Institución; también posee una interfaz poco intuitiva para el operario la cual hace que el proceso de emisión de licencias sea lento y engorroso, además de no poseer las validaciones necesarias para los datos, lo cual hace que los procesos de revisión manual retrasen todo tipo de emisión de licencias.

El problema anterior orienta la solución hacia el desarrollo de un nuevo Sistema de Gestión de Licencias del Personal Aeronáutico del IACC, que garantice la calidad y eficacia del proceso, en cumplimiento de los requisitos que actualmente se demandan, entre los que se encuentran: permitir la gestión de las habilitaciones de los titulares, generar certificados a miembros de la tripulación, generar certificados de convalidación a titulares externos, generar notificaciones, reportes y realizar la renovación de las licencias.

Teniendo en cuenta la situación problemática explicada se plantea el siguiente **problema de investigación:** ¿Cómo optimizar la gestión de Licencias del Personal Aeronáutico del Instituto de Aeronáutica Civil de Cuba?

Se define como **objetivo general:** desarrollar un sistema informático que permita llevar a cabo la gestión y emisión de los documentos de licencias del personal que se realiza en la Oficina de Licencias del Instituto de Aeronáutica Civil de Cuba.

Para dar solución a la problemática planteada el **objeto de estudio** está conformado por los procesos de gestión y emisión de documentos de licencias, del personal del Instituto Aeronáutico Civil de Cuba. Según lo planteado el **campo de acción** lo

constituyen los procesos de gestión y emisión ejecutados en la Oficina de Licencias del Instituto Aeronáutico Civil de Cuba.

Preguntas científicas

1. ¿Cuáles son los presupuestos teóricos que sustentan la implementación del Sistema de gestión de licencias del Instituto de Aeronáutica Civil de Cuba?
2. ¿Qué fundamentos presenta el análisis y diseño del Sistema de gestión de licencias del Instituto de Aeronáutica Civil de Cuba?
3. ¿Qué características debe tener la implementación de un sistema de gestión de licencias aeronáuticas?
4. ¿Cómo validar la contribución del Sistema de gestión de licencias del IACC para mejorar el proceso de emisión de documentos de licencias aeronáuticas en el IACC?

Objetivos específicos:

1. Investigar y realizar el diseño teórico de la investigación.
2. Definir las tecnologías, las herramientas y la metodología de desarrollo para la implementación.
3. Diseñar el Sistema de Gestión de Licencias del Personal Aeronáutico del Instituto de Aeronáutica Civil de Cuba.
4. Implementar el Sistema de Gestión de Licencias del Personal Aeronáutico del Instituto de Aeronáutica Civil de Cuba.
5. Validar la solución del Sistema de Gestión de Licencias del Personal Aeronáutico del Instituto de Aeronáutica Civil de Cuba.

Para alcanzar dichos objetivos se plantea desarrollar las siguientes **tareas de investigación:**

1. Realización del estudio del arte para documentar los diferentes sistemas que se utilizan para la generación de licencias en el ámbito nacional e internacional.
2. Selección de las tecnologías, herramientas y estándares que se necesitan para implementar el Sistema de Gestión de Licencias.
3. Selección de la metodología de desarrollo de *software* que guíe el desarrollo del Sistema de Gestión de Licencias del Personal Aeronáutico del Instituto de Aeronáutica Civil de Cuba.
4. Definición y especificación de los requisitos del *software* para adecuar el desarrollo del sistema a las exigencias del cliente.
5. Descripción de la arquitectura para el desarrollo más adecuado del sistema.

6. Ajuste de la solución según los patrones de diseño más adecuados para la usabilidad del sistema de generación de licencias para pilotos.
7. Implementación de los requisitos especificados en la programación del sistema.
8. Definición de las pruebas a realizar al sistema para validar su funcionamiento.
9. Validación de la solución implementada mediante pruebas de funcionalidad.

Métodos teóricos:

Analítico-sintético: se consultará la bibliografía necesaria para dar cumplimiento a las tareas de la investigación y se realizará un resumen de los principales aspectos de cada una de ellas.

Análisis Histórico - Lógico: se utilizará con el objetivo de indagar sobre el avance alcanzado por los sistemas de gestión de personal y generadores de licencias.

Métodos empíricos:

Entrevista: Se utilizará para dialogar con el cliente en los diferentes encuentros que se realicen para definir las funcionalidades, restricciones y las interfaces del módulo.

El presente trabajo está dividido en 3 capítulos, que recogen los temas abordados en la investigación y desarrollo del mismo:

Capítulo 1. Fundamentación teórica. En este capítulo se relacionan los elementos teóricos fundamentales que sustentan la presente investigación, para dar un conocimiento básico y una idea del sistema a desarrollar; se definen las herramientas y tecnologías a utilizar para llevar a cabo el desarrollo del sistema, teniendo en cuenta las necesidades a cumplir.

Capítulo 2. Análisis y diseño de la propuesta de solución. En este capítulo se muestra la propuesta solución y se hace un análisis detallado de su funcionamiento. Se realiza el levantamiento de requisitos funcionales y no funcionales, se chequean los principales aspectos respecto al diseño de la interfaz y se define la arquitectura completa del proyecto.

Capítulo 3. Implementación y prueba. En este capítulo se muestran los resultados de la propuesta de solución y la implementación llevada a la práctica. Se muestran los resultados obtenidos de las pruebas realizadas al sistema para comprobar el correcto funcionamiento de la misma.

Finalmente se presentan las **Conclusiones** y **Recomendaciones** derivadas de la investigación, las **Referencias Bibliográficas**, así como los **Anexos** que apoyan la comprensión y dan información adicional sobre el trabajo realizado.

Capítulo 1. Fundamentación teórica.

1.1 Introducción

El objetivo fundamental de este capítulo es hacer un análisis sobre el estado del arte de algunos sistemas homólogos actualmente funcionales. Se analiza y define la metodología a utilizar durante el desarrollo del sistema y se hace un estudio de las diferentes herramientas y tecnologías que pueden ser adecuadas para su desarrollo e implementación.

1.2 Conceptos fundamentales

Licencia: constituye un permiso para ejercer una determinada función. El término también permite nombrar al documento o contrato en que consta la licencia en cuestión. (Porto, y otros, 2014)

Habilitación: autorización inscrita en una licencia o asociada con ella, y de la cual forma parte, en las que se especifican condiciones especiales, atribuciones o restricciones referentes a dicha licencia (OACI, 2006). Las Habilitaciones están conformadas por Clases y Tipos. Las Clases son la autorización que un piloto debe obtener para poder volar una clase genérica de aeronave, permitiendo volar múltiples modelos de avión con la misma habilitación, mientras que el tipo es la autorización que un piloto debe obtener para poder volar un modelo específico.

Licencia aeronáutica: documento que avala a una persona como personal aeronáutico.

Personal aeronáutico: comprende a quienes desarrollan una actividad habitual y organizada, como factor de la navegación aérea, tanto a bordo de aeronaves en vuelo, como en los servicios de infraestructura y ayuda directa a dicha navegación desde tierra. Se excluyen, por tanto, del concepto, el personal que presta sus servicios en empresas o industrias aeronáuticas (ingenieros, médicos, economistas, abogados, entre otros), como el que, en los aeródromos y aeropuertos, realiza trabajos auxiliares que no requieren una cualificación específicamente aeronáutica.

Dentro del personal aeronáutico se distingue el personal que realiza funciones técnico-aeronáuticas (el comandante y pilotos de la aeronave, mecánicos y radiotelegrafista) y los que realizan funciones auxiliares no aeronáuticas (sobrecargo y azafatas). Por lo que se refiere al personal aeronáutico de superficie, debe destacarse el personal encargado de los servicios de ayuda, control e información de la navegación aérea (controladores, encargados de operaciones de vuelo, oficiales de tráfico, encargados de

estaciones aeronáuticas y meteorología), así como los jefes de aeródromo y aeropuerto. (Enciclopedia jurídica, 2014)

Restricción: en el caso de la aeronáutica se le denomina restricciones al conjunto de requisitos necesarios para solicitar una licencia aeronáutica. Es obligatorio que el solicitante cumpla con todas las restricciones especificadas, para el tipo de licencia solicitada.

1.3 Sistemas de gestión de licencias

Los sistemas de gestión son herramientas que permiten optimizar los recursos, reducir costes y mejorar la productividad una empresa o instituto. Estos instrumentos de gestión reportan datos en tiempo real que permiten tomar decisiones para corregir fallos y prevenir la aparición de gastos innecesarios. Los sistemas de gestión están basados en normas internacionales que permiten controlar distintas facetas en una empresa, como la calidad de su producto o servicio, los impactos ambientales que pueda ocasionar, la seguridad y salud de los trabajadores, la responsabilidad social o la innovación. (integra, 2016)

Los sistemas de gestión de licencias aeronáuticas están orientados a mejorar los procesos de las instituciones y departamentos aeronáuticos. Estos están enfocados principalmente al otorgamiento de licencias aeronáuticas y habilitaciones al personal aeronáutico. Dichos sistemas deben estar regidos por los parámetros establecidos por la OACI, en cuanto a habilidades necesarias según el tipo de licencia a solicitar y la labor que realizará, así como los requisitos imprescindibles que debe cumplir la persona.

1.4 Sistemas Homólogos

El uso de sistemas de gestión de licencias aeronáuticas es imprescindible en la actualidad al facilitar el trabajo y la seguridad en las aerolíneas. A continuación, se describen algunos y las principales características que presentan.

1.4.1 A nivel internacional

Muchas agencias de vuelo hacen uso de las tecnologías a través de *Internet* para la solicitud de licencias de pilotos, como es el caso de los sitios *web* de la DGAC (Dirección General de Aeronáutica Civil) de Guatemala y de la AESA (Agencia Estatal de Seguridad Aérea) de España; el primero hace referencia a los requisitos para solicitar la licencia y a dónde remitirse para realizar el trámite y el segundo publica cuestionarios online para hacer solicitudes de licencias o habilitaciones para los pilotos.

Sistema de Licencias Aeronáuticas de Chile (ALVI)

Es una plataforma *web* que la Dirección General de Aeronáutica Civil de ese país ha puesto a disposición de la comunidad aeronáutica. Fue creado especialmente en beneficio del personal aeronáutico y tiene como propósito el otorgamiento de licencias y habilitaciones. Los usuarios pueden enviar, monitorear y tramitar las solicitudes de licencias y/o habilitaciones a través de Internet e imprimir su licencia.

Destacan entre los beneficios del Sistema ALVI, el ahorro de tiempo, ingreso por ventanilla única, envío de solicitud vía Internet, digitalización de documentos, acceso controlado mediante clave, acceso a carpeta electrónica personal, seguimiento del trámite solicitado, captura e impresión de licencia desde cualquier lugar del mundo y el sistema de exámenes teóricos automatizado.

Los servicios van dirigidos a los pilotos, Tripulantes Auxiliar de Cabina, Operadores de Sistema, Ingenieros Aeronáuticos, Supervisores de Mantenimiento, Mecánicos, Controladores de Tránsito Aéreo, Encargados de Operaciones de Vuelo y los Operadores de Servicios de Vuelo.

Los solicitantes pueden obtener como productos el otorgamiento de licencias y habilitaciones, la revalidación de licencias y habilitaciones, la convalidación de licencias extranjeras, certificados y duplicados de licencias. (DIRECCIÓN GENERAL DE AERONÁUTICA CIVIL, 2016)

Sistema Informático del Personal Aeronáutico(SIPA) de Ecuador

En el 2014, la DGAC implementó el Sistema Informático del Personal Aeronáutico (SIPA), en la página *web* institucional, para que los pilotos, despachadores, tripulantes de cabina, mecánicos, controladores y alumnos pilotos, puedan aplicar desde cualquier lugar o condición geográfica al otorgamiento y/o habilitación de la licencia aeronáutica.

El SIPA es un sistema integrado diseñado para otorgar, renovar, actualizar y convalidar las licencias y habilitaciones del personal técnico aeronáutico civil. Permite al usuario ingresar en línea su información y documentación requerida para ser elegible a la licencia aeronáutica solicitada.

Una vez culminado este proceso el aspirante acude al Departamento de Evaluación de la DGAC a rendir, a través del SIPA, el examen de conocimientos teóricos y de forma inmediata conoce el porcentaje obtenido, cuyo puntaje mínimo es 75%. En caso que el aspirante falle en el examen, puede aplicar a uno nuevo de acuerdo a lo establecido en la regulación técnica de aviación civil. Para la prueba, el solicitante cuenta con un banco de preguntas específicas para cada tipo de licencia aeronáutica.

Con la implementación del sistema SIPA los tiempos de entrega de la licencia aeronáutica se redujeron de una semana a un día e incluso horas, mejorando así los

niveles óptimos de servicio a los usuarios de la DGAC. (Direccion General de Aviación Civil, 2015)

1.4.2 A nivel nacional

Sistema para el control de licencias y habilitaciones

El Sistema para el control de licencias y habilitaciones, es actualmente usado por la Oficina de Gestión de Licencias del IACC. Este sistema fue programado en Delphi y utiliza Access 2003 como gestor de base de datos. Tiene como objetivo, llevar el control exhaustivo de la información del personal de vuelo, lo cual permite realizar todo lo referente a la emisión, habilitación, prórroga y renovación de licencias, así como búsquedas avanzadas por disímiles criterios y estadísticas.

Dentro de sus principales opciones se encuentran:

- Cuenta con un menú principal en el que están contenidas las opciones del sistema.
- Cuenta con una barra de herramientas donde se encuentran contenidas las opciones, al igual que el menú.
- Mediante las preferencias se pueden configurar los parámetros de trabajo, tales como, el directorio donde se encuentran las bases de datos y el camino donde se realizará la salva.
- Cuenta con una pantalla de nomencladores mediante las cuales se puede actualizar y personalizar la información del sistema.
- Cuenta con una opción para la renovación.
- Cuenta con una opción para la habilitación.
- Cuenta con una opción para la convalidación.
- Cuenta con una opción para la iniciación.
- Cuenta con una opción para la prórroga.
- Todas las opciones anteriores culminan con la impresión de la licencia oficial, estipulada para las autoridades competentes.
- Se pueden hacer cualquier tipo de consultas y búsquedas, así como reportes personalizados.
- Se pueden obtener estadísticas sobre cualquier tipo de información de forma rápida.

Después de un análisis de dichos sistemas, se decide no adoptar ninguna de las soluciones existentes en el ámbito internacional, dadas las diferencias en el

funcionamiento de las empresas en cuestión y que, aun cumpliendo con los requisitos necesarios, no poseen las especificidades adecuadas al país. En cuanto a los sistemas a nivel nacional, tomando en consideración las insuficiencias que presenta el actual sistema generador de licencias, se determina tomarlo solamente como referencia y desarrollar un nuevo sistema con herramientas más modernas y ajustado a las necesidades actuales del cliente.

1.5 Metodologías de Desarrollo de *Software*

En el ámbito de la informática que está centrado hacia los procesos de desarrollo, se busca siempre el modo de trabajar eficientemente para evitar descoordinaciones que han llevado a un elevado número de proyectos a terminar sin éxito. El objetivo de un proceso de desarrollo es aumentar la calidad del *software* (en todas las fases por las que transita) a través de una mayor transparencia y control sobre el proceso. Por ello es de gran interés analizar algunas metodologías de desarrollo que se pueden utilizar para llevar a cabo un software sin importar el tipo que sea.

Existen dos tipos de metodologías, las ágiles y las tradicionales.

Metodologías Ágiles

Son las metodologías que permiten incorporar cambios con rapidez en el desarrollo de *software*. En muchas ocasiones, los modelos de gestión tradicionales no sirven para afrontar un reto que hoy en día resulta fundamental, incorporar cambios con rapidez y en cualquier fase del proyecto.

Se trata de evitar lo que tantas veces ha ocurrido, cuando el proyecto se encuentra bastante avanzado y no va por el buen camino o, simplemente, el cliente decide introducir cambios sustanciales y esos cambios obligan a desechar todo el trabajo realizado hasta entonces, lo que impide acabar en el plazo previsto. Dado que los cambios nunca van a dejar de existir, es necesario ser capaces de gestionar los proyectos de una forma más ágil. Las metodologías ágiles más destacadas son *XP*, *AUP*, *Scrum*, *Iconix,* y *Cristal Methods.*

XP

Es una metodología ágil para el desarrollo de software y consiste básicamente en ajustarse estrictamente a una serie de reglas que se centran en las necesidades del cliente para lograr un producto de buena calidad en poco tiempo, centrada en potenciar las relaciones interpersonales como clave para el éxito del desarrollo de software.
La filosofía de XP es satisfacer al completo las necesidades del cliente, por eso lo integra como una parte más del equipo de desarrollo. Promueve el trabajo en equipo,

preocupándose en todo momento del aprendizaje de los desarrolladores y estableciendo un buen clima de trabajo. Este tipo de programación es la adecuada para los proyectos con requisitos imprecisos, muy cambiantes y donde existe un alto riesgo técnico.

XP está diseñada para el desarrollo de aplicaciones que requieran un grupo de programadores pequeño, dónde la comunicación sea más factible que en grupos de desarrollo grandes. La comunicación es un punto importante y debe realizarse entre los programadores, los jefes de proyecto y los clientes. (Borja, 2016)

SCRUM

Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos años. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas *sprints*, con una duración de 30 días. El resultado de cada *sprint* es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto. Éstas son las verdaderas protagonistas, especialmente la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración. (Letelier, y otros, 2010)

Crystal Methodologies

Se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo (de ellas depende el éxito del proyecto) y la reducción al máximo del número de artefactos producidos. El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores, por ejemplo Crystal Clear (3 a 8 miembros) y Crystal Orange (25 a 50 miembros). (Letelier, y otros, 2010)

AUP-UCI

La UCI desarrolló una versión de la metodología de desarrollo ágil de *software AUP*, de forma tal que se adapte al ciclo de vida definido para su actividad productiva. Esta versión decide mantener para el ciclo de vida de los proyectos la fase de Inicio, pero modifica el objetivo de la misma y se unifican las restantes fases de la metodología de desarrollo de *software* AUP en una sola, denominada Ejecución, y agregándose también una nueva fase denominada Cierre. (Sánchez, 2015)

Metodologías Tradicionales

Las metodologías tradicionales se focalizan en documentación, planificación y procesos. Su atención va dirigida a llevar una documentación exhaustiva de todo el proyecto y a cumplir con el plan que se traza para la confección del *software*, definiendo todo esto en la fase inicial del desarrollo del proyecto. Otra característica importante de este enfoque es que se tienen altos costos al implementar un cambio lo que no garantiza una buena solución para proyectos donde el entorno es volátil. Algunos ejemplos de metodologías tradicionales son: *RUP, MSF, Iconix, y Win-Win Spiral Model*. (Téllez, 2012) (Instituto Tecnológico de Toluca, 2009)

Para un mayor entendimiento sobre el tema se presenta la siguiente tabla comparativa sobre los dos tipos de metodologías. (Avante, 2013)

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparadas para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo de desarrollo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del *software*	La arquitectura del *software* es esencial y se expresa mediante modelos

Tabla 1 Diferencia entre tecnologías ágiles y tradicionales. Fuente, (Avante, 2013)

Por todo lo anteriormente planteado se concluye que es más apropiado para este proyecto utilizar una metodología ágil, por lo que se selecciona la metodología *AUP-UCI*, que se adapta al contexto de la universidad.

1.6 Servicios *web*

La necesidad de estandarizar la comunicación entre diversas plataformas y lenguajes de programación provoca el surgimiento de los Servicios *web*, estos no son más que un conjunto de estándares y protocolos.

El sistema a desarrollar generará un servicio *web* en código *PHP*, pues inicialmente se pretende que esta sea usada por aplicaciones desarrolladas en este lenguaje. En la creación del servicio se tuvo en cuenta el uso de *PDO* (PHP Data Objects), este es una extensión de capa de abstracción que permite obtener datos y realizar consultas de igual forma para distintos gestores de base de datos. (Cardentey, 2009)

La ventaja principal que aporta el uso de estos servicios para el desarrollo de este trabajo, es que permite a la aplicación desplegar un ambiente diferente, que pueda intercambiar información con los orígenes de datos existentes, sin importar las plataformas en que estos se encuentren o sus propiedades. Permite el acceso desde cualquier lugar del mundo, solo es necesario conexión a internet y un navegador *web.* No necesita instalación. Se garantiza el control absoluto sobre los usuarios. Es fácil de actualizar, los usuarios siempre utilizan la versión más actualizada con el último lanzamiento y también pueden ser utilizadas por múltiples usuarios al mismo tiempo. (Internet Ya, 2017)

1.7 Análisis de las tecnologías, herramientas y lenguajes a utilizar

1.7.1 Lenguajes

Procesador de Hipertexto (PHP)

PHP es un lenguaje de programación interpretado de alto nivel que se ejecuta en un servidor *web* y permite la generación de páginas dinámicas. Puede ser desplegado en la mayoría de los servidores *web* y en casi todos los sistemas operativos y plataformas sin costo alguno. Al ser un lenguaje libre dispone de una gran cantidad de características que lo convierten en la herramienta ideal para la creación de páginas *web* dinámicas. Características fundamentales del lenguaje *PHP*:

- Soporta de conexiones a bases de datos tales como: *MySQL*, *PostgreSQL*, *Oracle*, *Informix*, entre otras.
- Interactúa con uno de los servidores de *web* más potentes como el *Apache*.
- Se integra con varias bibliotecas externas, permite generar documentos y analizar código *XML*.
- Utiliza las sesiones de *HTTP*, conectividad de *Java*, *LDAP*, *SNMP* e *IMAP*.

- Tiene manejo de excepciones (desde PHP5).
- No se restringe a una metodología para programar, el programador puede utilizar cualquier técnica de desarrollo que le permita escribir el código de forma ordenada, estructurado y adaptable.
- Las principales características de *PHP* son: rapidez; facilidad de aprendizaje; soporte multiplataforma tanto de diversos Sistemas Operativos, como servidores *HTTP* y de bases de datos.

(Guzmán, 2011)

Python

Python es un lenguaje de programación desarrollado como proyecto de código abierto y es administrado por la empresa Python Software Foundation.

Características generales de Python

1. Lenguaje de programación de alto nivel, diseñado para ser fácil de leer y simple de implementar.
2. Es código abierto.
3. Puede ejecutarse en *Mac*, *Windows* y sistemas *Unix*; también ha sido portado a máquinas virtual *JAVA* y *.NET*.
4. Es a menudo usado para desarrollar aplicaciones *web* y contenido *web* dinámico. Se utiliza para crear extensiones tipo *plug-ins* para programas de 2D y 3D como *Autodesk Maya*, *GIMP*, *Blender*, *Inkscape*, etc.
5. Los *scripts* de *Python* tienen la extensión de archivo .py, que pueden ser ejecutados inmediatamente.
6. Permite grabar programas compilados con extensión de archivo. pyc, los cuales suelen ser usados como módulo que pueden ser referenciados por otros programas *Python*. (ALEGSA, 2016)

UML

UML son las siglas de “Unified Modeling Language” o “Lenguaje Unificado de Modelado”. Se trata de un estándar que se ha adoptado a nivel internacional por numerosos organismos y empresas para crear esquemas, diagramas y documentación relativa a los desarrollos de software (programas informáticos).

El término “lenguaje” ha generado bastante confusión respecto a lo que es UML. En realidad, el término lenguaje quizás no es el más apropiado, ya que no es un lenguaje propiamente dicho, sino una serie de normas y estándares gráficos respecto a cómo se

deben representar los esquemas relativos al software. Mucha gente piensa por confusión que UML es un lenguaje de programación y esta idea es errónea: UML no es un lenguaje de programación. Como decimos, UML son una serie de normas y estándares que dicen cómo se debe representar algo.

Es una herramienta propia de personas que tienen conocimientos relativamente avanzados de programación y es frecuentemente usada por analistas funcionales (aquellos que definen qué debe hacer un programa sin entrar a escribir el código) y analistas-programadores (aquellos que, dado un problema, lo estudian y escriben el código informático para resolverlo en un lenguaje como Java, C#, Python o cualquier otro). Por tanto, si estás dando tus primeros pasos en programación, te recomendaríamos que te olvides de UML hasta que tengas unos conocimientos mínimos como uso de condicionales, bucles, y conocimiento de la programación orientada a objetos. Esto es solo una recomendación, en realidad prácticamente cualquier persona puede usar UML, incluso podría usarse para realizar esquemas o documentación de procesos que no tengan que ver con la informática. (aprenderaprogramar, 2017)

1.7.2 Herramientas

Servidor *web* Apache

Es un servidor *web* flexible, rápido y eficiente, continuamente actualizado y adaptado a las últimas versiones de protocolos. Es capaz de funcionar en la mayoría de las plataformas y entornos existente.

Está entre los servidores *web* más reconocidos a nivel internacional, debido a la capacidad de correr en múltiples *Sistemas Operativos*. *Apache* es una tecnología gratuita de código abierto y altamente configurable, de diseño modular capaz de interpretar diversos lenguajes de programación entre los que se encuentran *Perl* y *PHP*. También, permite personalizar la respuesta ante los posibles errores que pueden surgir en el servidor.

El servidor *Apache* está estructurado en módulos, los cuales pueden clasificarse en tres categorías específicas:

- **Módulos Base:** son funciones básicas del *Apache*.
- **Módulos Multiproceso:** responsables de la unión con los puertos de la máquina, aceptando las peticiones y enviando a los hijos a atender las peticiones.

- **Módulos Adicionales:** cualquier otro módulo que le añada una funcionalidad al servidor se puede añadir sin necesidad de volver a instalar el *software*.

(ESI, 2016)

PgAdmin III

Es una Herramienta de código abierto para la administración. De Bases de Datos *PostgreSQL*, incluye:

- Interfaz administrativa grafica
- Herramienta de consulta *SQL*
- Editor de código
- Agente de planificación SQL/Shell/batch. La interfaz gráfica soporta todas las características de *PostgreSQL* y hace simple la administración, disponible en varios Sistemas Operativos: *Windows, Linux, FreeBSD, Mac OSX y Solari.*

(Espino, 2017)

1.7.3 Gestores de Base de datos

PostgreSQL

Es un Sistema de Gestión de Bases de Datos Objeto-Relacionales (ORDBMS) de código abierto. Comenzó como un proyecto en la Universidad Berkeley de California y actualmente ofrece control de concurrencia multi-versión, soportando casi toda la sintaxis *SQL* incluyendo subconsultas, transacciones y funciones definidas por el usuario. Proporciona un gran número de características que sólo se encontraban en las bases de datos comerciales tales como *DB213* y *Oracle14*. Ventajas fundamentales de *PostgreSQL*:

- **Extensible:** el código fuente se encuentra disponible para todos sin costo alguno. Si se necesitara extender o personalizar, pueden hacerlo con un mínimo esfuerzo, sin costos adicionales.
- **Multiplataforma:** está disponible para cualquier sistema *Unix15* y una versión nativa de *Windows* que actualmente se encuentra estado beta de pruebas.
- **Diseñado para ambientes de alta concurrencia:** PostgreSQL usa una estrategia de almacenamiento de filas llamada MVCC (Multiversión de Control de Concurrencia) para conseguir una respuesta más eficiente en ambientes de grandes volúmenes. Los principales proveedores de sistemas de bases de datos comerciales usan también esta tecnología, por las mismas razones.

- **Alta concurrencia:** mientras un proceso escribe en una tabla, otros accedan a la misma sin necesidad de bloqueos.
- **Integridad de los datos:** claves primarias, llaves foráneas con capacidad de actualizar en cascada o restringir la acción y restricción no vacía.
- **Resistencia a fallas:** escritura de registros anticipada para evitar pérdidas de datos en caso de fallos por: Energía, Sistema Operativo, *hardware*.

(TiendaLinux, 2016)

MySQL

MySQL es el sistema de gestión de base de datos relacional más popular de código abierto del mundo. Se desarrolla y se distribuye con el apoyo de Oracle Corporation. Su arquitectura lo hace extremadamente rápido, fiable, fácil de usar y adaptar. (Pena Aponte, 2014)

Entre sus ventajas se pueden encontrar:

- Bajo costo en requerimientos para la elaboración de bases de datos, ya que debido a su bajo consumo puede ser ejecutado en una máquina con escasos recursos sin ningún problema.
- Facilidad de configuración e instalación.
- Soporta gran variedad de sistemas operativos.
- Baja probabilidad de corromper datos, incluso si los errores no se producen en el propio gestor, sino en el sistema en el que está.

(Sribd, 2016)

Microsoft SQL Server

Microsoft SQL Server es un sistema de manejo de bases de datos para la administración del modelo relacional, desarrollado por la empresa Microsoft.

Dentro de los competidores más destacados de *SQL Server* están: *Oracle*, *MariaDB*, *MySQL*, *PostgreSQL*.

SQL Server solo está disponible para sistemas operativos Windows de Microsoft.

Puede ser configurado para utilizar varias instancias en el mismo servidor físico, la primera instalación lleva generalmente el nombre del servidor, y las siguientes - nombres específicos (con un guion invertido entre el nombre del servidor y el nombre de la instalación). (Moreno, 2017)

1.7.4 Framework

Symfony

Symfony es un *framework* diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones *web*. Para empezar, separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación *web*. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación *web* compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación.

Symfony está desarrollado completamente con *PHP*. Es compatible con la mayoría de gestores de bases de datos, como *MySQL*, *PostgreSQL*, *Oracle* y *SQL Server* de Microsoft. Se puede ejecutar tanto en plataformas **nix* (*Unix, Linux,* etc.) como en plataformas *Windows*. A continuación, se muestran algunas de sus características.

Características de *Symfony*:

- Fácil de instalar y configurar en la mayoría de plataformas (y con la garantía de que funciona correctamente en los sistemas Windows y **nix* estándares).
- Independiente del sistema gestor de bases de datos.
- Sencillo de usar en la mayoría de casos, pero lo suficientemente flexible como para adaptarse a los casos más complejos.
- Basado en la premisa de "convenir en vez de configurar"*,* en la que el desarrollador solo debe configurar aquello que no es convencional.
- Sigue la mayoría de mejores prácticas y patrones de diseño para la *web.*
- Preparado para aplicaciones empresariales y adaptable a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.
- Código fácil de leer que incluye comentarios de *phpDocumentor* y que permite un mantenimiento muy sencillo.
- Fácil de extender, lo que permite su integración con librerías desarrolladas por terceros.

(symfony, 2017)

Django

Es un *framework* que ahorra tiempo considerable en el desarrollo *web*, su filosofía es bajo acoplamiento, menos código enfocándose en el desarrollo rápido y explícito.

Django presenta uno de los sistemas de plantillas más flexible y potente existente, separando la lógica de la presentación, desalentando la redundancia, permitiendo la seguridad y extensibilidad, así como el fácil desacoplo del código *HTML*, el sistema de plantilla de *Django* es ideal para los diseñadores, no para programadores, como es el caso de *web.py* o *PHP*; en cambio nos permite insertar algunas sentencias del lenguaje, así como la creación de nuestras propias extensiones.

Una de las principales ventajas de *Django*, es que no se entromete, dejándo trabajar fuera del ámbito del framework según se necesite. Otras de sus ventajas son: su potencia, infinita flexibilidad, encierra las mejores prácticas, diferencia el *GET* del *POST*, eficiencia *SQL*, sintaxis concisa y poderosa, opción de escribir *SQL* crudo cuando se necesita, bajo acoplamiento en el diseño *URLs* definitivas, uso de objetos *request* y permite el desarrollo en equipo. (Doutdeslibretas, 2011)

1.7.5 Entorno de desarrollo integrado(IDE)

PyCharm

PyCharm es un entorno de desarrollo para los desarrolladores de *Python* y *Django*, presenta un editor de códigos inteligente, que entiende los detalles específicos de *Python* y ofrece extraordinarios mejoradores de la productividad entre ellos, formateo automático de código, finalización de código, refactorizaciones, importación automática, navegación de código con un solo clic. Respaldadas por avanzadas rutinas de análisis de código, estas características hacen de *PyCharm* una poderosa herramienta tanto en las manos de los desarrolladores *Python* profesionales como en las de quienes recién comienzan a usar la tecnología. (PR Newswire, 2010)

Php Storm

JetBrains PhpStorm es un *IDE* comercial multiplataforma para *PHP* construido sobre la plataforma *IntelliJ IDEA* de *JetBrains*. *PhpStorm* proporciona un editor para *PHP*, *HTML* y *JavaScript* con análisis de código al instante, prevención de errores y refactorizaciones automatizadas para *PHP* y código *JavaScript*. La terminación de código de *PhpStorm* soporta *PHP* 5.3, 5.4, 5.5, 5.6 & 7.0 incluyendo generadores. Incluye un editor *SQL* completo con resultados de consulta editables. *PhpStorm* se basa en *IntelliJ IDEA*, que está escrito en *Java*. Los usuarios pueden extender el *IDE* mediante la instalación de complementos creados para la plataforma *IntelliJ* o escribir sus propios complementos. Todas las características disponibles en *WebStorm* se incluyen en *PhpStorm*, que añade

soporte para *PHP* y bases de datos. *WebStorm* se suministra con complementos de *JavaScript* preinstalados (como para Node.js), que están disponibles para *PhpStorm*. (PR Newswire, 2011)

1.7.6 Herramientas Case

Visual Paradigm

Visual Paradigm es una herramienta para desarrollo de aplicaciones utilizando modelado *UML* (Lenguaje Unificado de Modelado) ideal para Ingenieros de *Software*, Analistas de Sistemas y Arquitectos de sistemas que están interesados en construcción de sistemas a gran escala y necesitan confiabilidad y estabilidad en el desarrollo orientado a objetos. (Targetware Informática S.A.C., 2017)
Visual Paradigm también ofrece:

- Navegación intuitiva entre la escritura del código y su visualización Potente generador de informes en formato *PDF/HTML*.
- Ambiente visualmente superior de modelado.
- Sincronización de código fuente en tiempo real.

Rational Rose Enterprise

Rational Rose es una herramienta CASE desarrollada y mantenida por *Rational Corporation*. Está orientada a objetos sobre la base de UML, facilita el proceso de modelado con un número significativo de estereotipos predefinidos y permite, además, la generación de la documentación del *software* (De Nobrega). Esta herramienta posee un mecanismo para generar el código de las clases definidas en el diseño UML, aunque no soporta varios lenguajes de programación.

1.7.7 Selección de las tecnologías

Luego de un análisis entre las herramientas más usadas en el ámbito de la informática actual para el desarrollo de aplicaciones *web*, se seleccionan las que se consideraran más apropiadas para la elaboración del sistema, que garantizan tanto agilidad en el proceso, como la calidad final de este.

Se seleccionó el lenguaje PHP debido a que posee ventajas notables respecto a otros lenguajes en cuanto a rapidez y facilidad de aprendizaje. Además, es un lenguaje multiplataforma, es decir, funciona en diferentes Sistemas Operativos. Debido a la

elección de este lenguaje **es seleccionado el IDE PhpStorm 2016.2.2** por ser una potente herramienta para el desarrollo de aplicaciones basadas en PHP

Se seleccionó el *servidor web Apache,* por ser una herramienta gratuita, de código abierto, poseer capacidad de funcionar en la mayoría de las plataformas y entornos existente, además de poseer una alta rapidez y eficiencia comparado con otros servidores *web*.

El gestor de Base de Datos seleccionado es el PostgresSQL 9.4. Es la opción que más se ajusta al sistema y ha sido utilizado anteriormente en disímiles situaciones demostrando ser una herramienta potente en cuanto a su alta concurrencia, integridad de los datos y resistencia a fallas.

Se seleccionó el PgAdmin III por ser la herramienta de preferencia en la UCI para trabajar con el gestor de Base de Datos de *Postgres*.

No existen muchas diferencias entre *Visual Paradigm* y *Rational Rose* en cuanto a las funcionalidades básicas necesarias, por lo que **se seleccionó *Visual Paradigm 8,*** teniendo en cuenta que es la herramienta con la que más experiencia cuenta el equipo de trabajo por concepto de uso.

Debido a que el lenguaje seleccionado fue el *PHP* es recomendable **utilizar el *framework Symfony***, ya que el mismo es un *framework* basado en *PHP*.

1.8 Conclusiones parciales

El análisis de las características y funcionamiento de algunos sistemas homólogos creados internacionalmente, así como los existentes en el país, demostraron la necesidad de tomar el sistema actual solo como referencia y desarrollar uno nuevo denominado “Sistema de Gestión de Licencias al Personal Aeronáutico del Instituto de Aeronáutica Civil de Cuba” que optimice el proceso en cumplimiento de los requerimientos del cliente.

El análisis de *Visual Paradigm* como herramienta *CASE* para modelar *UML*, *PHP* como lenguaje de programación, *PhpStorm* como *IDE* de desarrollo, para apoyar el ciclo de vida de un *software* guiado por un enfoque ágil con la metodología *AUP-UCI*, demostró las potencialidades de todas estas tecnologías, en aras de lograr la creación del sistema propuesto en la presente investigación de acuerdo a sus necesidades específicas.

Capítulo 2. Análisis y diseño de la propuesta de solución.

2.1 Introducción

En este capítulo se abordan aspectos fundamentales relacionados con el diseño del sistema a desarrollar. Entre los elementos a destacar se encuentran el diagrama del modelo del dominio, mediante el cual se representan las clases conceptuales significativas del problema a resolver; los principales requerimientos funcionales y no funcionales que se tuvieron en cuenta para la realización de la propuesta de solución; algunos patrones de diseño utilizados para lograr buenas prácticas de diseño y programación y se explica la estructura de la base de datos desarrollada.

2.2 Modelo del dominio

La modelación del dominio constituye una herramienta de apoyo para garantizar la descripción de las clases o conceptos y sus relaciones más importantes dentro del contexto del problema y permite una mejor comprensión del problema en cuestión, en este caso el actual sistema generador de licencias.

Conceptos	Descripción
Licencia(aeronáutica)	Documento oficial que acredita a la persona como piloto.
Examen médico	Resultado del examen médico el cual dicta si la persona está saludable para desarrollar la función
Vence Médico	Fecha de vencimiento del examen médico
Fecha solicitud	Fecha en la cual se realiza la solicitud de la licencia
Fecha vencimiento	Fecha en la cual expira la validez de la licencia
Examen Competencia	Indica la puntuación en el examen teórico.
Vence Técnico	Fecha en la cual el piloto debe volver a realizar el examen teórico
Solicitar	Solicitar una licencia a un personal que no posee ninguna
Prorrogar	Prorroga la fecha de creación de la licencia con el objetivo de prorrogar así la fecha de vencimiento de la misma
Convalidar	Hace válida la licencia, con el respaldo del examen teórico realizado.
Renovar	Renueva una licencia vencida.

Añadir Habilitación	Añade una habilitación, las habilitaciones están conformadas por clases y tipos.
Tipo	Un tipo se refiere a la función aeronáutica realizada. Ejemplo: piloto, copiloto, alumno piloto, instructor.
Clase	Una clase comprende un grupo determinado de aeronaves. Ejemplo: monomotor, hidroavión, helicóptero

Tabla 2 Conceptos del modelo del dominio

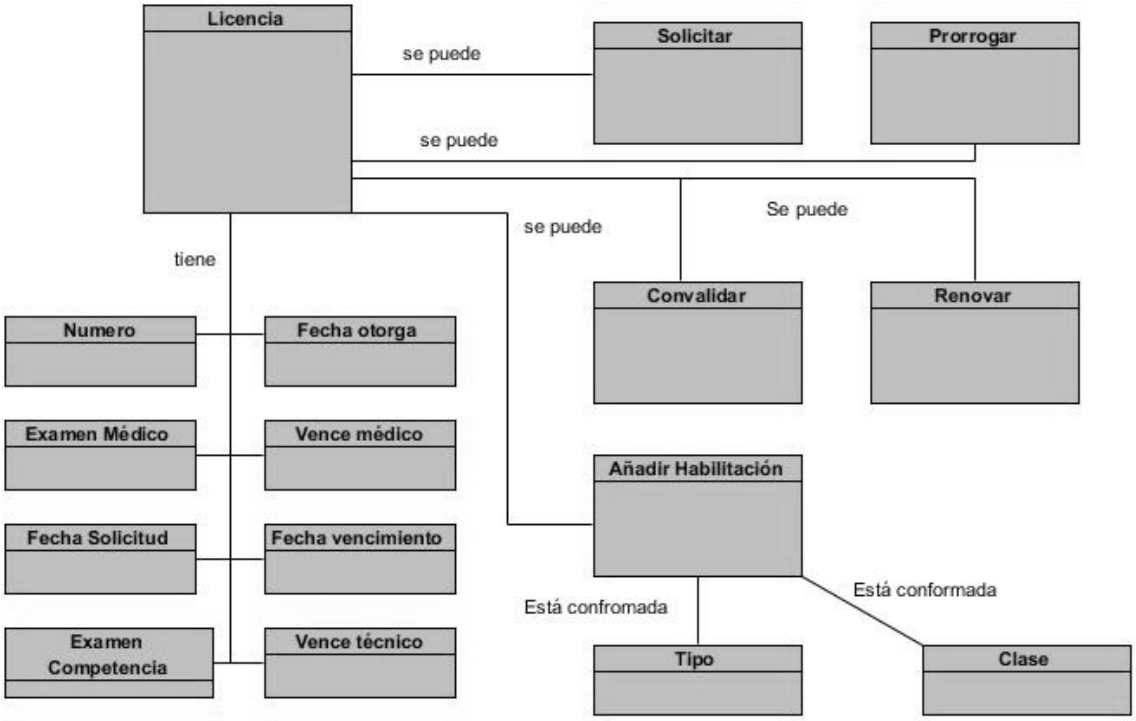


Imagen 1 Diagrama Modelo del dominio

2.3 Propuesta de Solución

Se desarrollará un sistema *web,* el cual se usará por los operarios establecidos. Para acceder al sistema se necesitará un usuario y una contraseña válida y se podrá añadir nuevos usuarios por el usuario administrador. Se permitirá gestionar los titulares, titulares externos y miembros de tripulación del personal aeronáutico. Se permitirá adicionar habilitaciones a un titular, así como gestionar los distintos tipos de habilitaciones existentes. Serán incluidas nuevas funcionalidades como convalidar licencias. Será posible desde el sistema la exportación de licencias, certificados de validez y certificados a miembros de la tripulación. Se podrá gestionar los diferentes centros de instrucción, programas de instrucción y simuladores de vuelo con que cuenta la agencia. Se generarán notificaciones, además reportes en cuanto a horas de vuelo de un tripulante técnico por períodos de tiempo por avión y a trámites realizados

mensual y anualmente.A continuación, se muestra la **imagen 2,** la cual explica el flujo principal del negocio y los principales cambios en la estructura en comparación con el anterior sistema

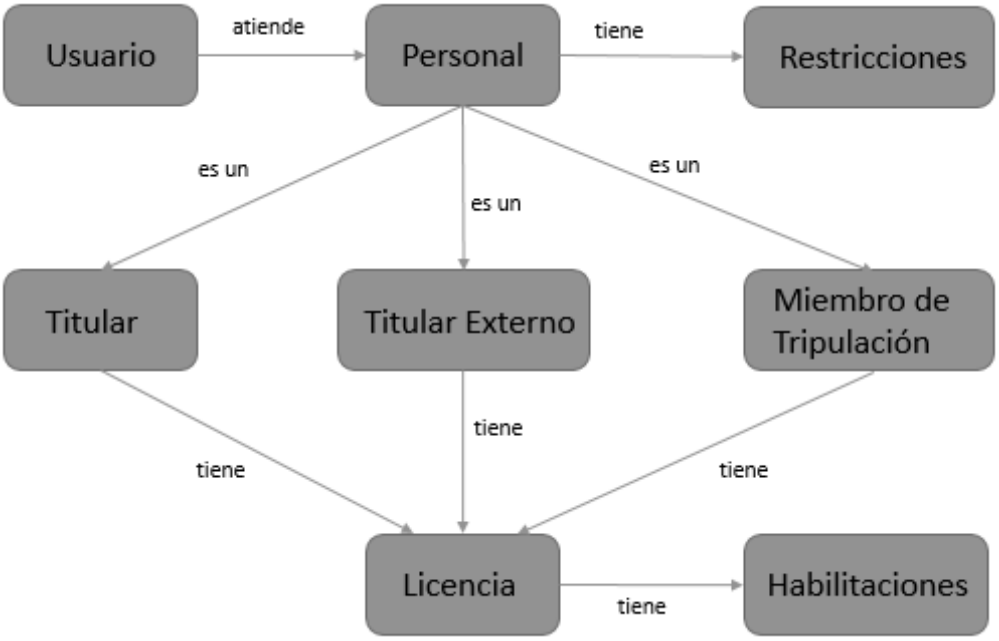


Imagen 2 Propuesta de Solución

2.4 Especificación de los Requisitos del *Software*

En la ingeniería del *software*, en la etapa de diseño del producto, los requisitos se utilizan como datos de entrada y establecen qué debe hacer el sistema, pero no cómo hacerlo. Son una condición o capacidad que un usuario necesita para poder resolver un problema o lograr un objetivo. De manera general estos requisitos son lo que el sistema debe hacer o una cualidad que el sistema debe poseer (Sommerville, 2012).

La definición de requisito en la literatura científica cuenta con varias acepciones; según la IEEE en ISO 2011 enuncia el concepto de la siguiente manera: “Declaración que se traduce o expresa una necesidad y sus limitaciones y las condiciones correspondientes.”

Es de suma importancia la correcta captura de requisitos, pues la mayoría de los errores en los proyectos se originan en el diseño preliminar y en una mala especificación de requisitos (Méndez, 2009)

Durante el desarrollo de la investigación se utilizaron una serie de técnicas para la captura de requisitos las cuales son (Monsalve, 2010):

Tormenta de ideas

Las tormentas de ideas (del inglés: brainstorming) Es una técnica de reuniones en grupo, se caracteriza por ser sencilla y fácil de aplicar. El objetivo de la misma es que los participantes muestren sus ideas en un ambiente libre de críticas o juicios. Consiste en la mera acumulación de ideas y/o información sin evaluar las mismas. Las tormentas de ideas suelen ofrecer una visión general de las necesidades del sistema, pero normalmente no sirve para obtener detalles concretos del mismo, por lo que suele aplicarse en los primeros encuentros.

Reunión con el cliente

Resulta una técnica muy aceptada dentro de la ingeniería de requisitos y su uso está ampliamente extendido. A través de esta técnica el equipo de trabajo se acerca al problema de una forma natural y le permite comprender los objetivos de la solución buscada.

Prototipado

Algunas propuestas se basan en obtener de la definición de requisitos prototipos que, sin tener la totalidad de la funcionalidad del sistema, permitan al usuario hacerse una idea de la estructura de la interfaz del sistema con el usuario. Un prototipo en *software* es un modelo del comportamiento del sistema que puede ser usado para entenderlo completamente o ciertos aspectos de él y así clarificar los requisitos. (Suarez, y otros, 2013)

2.4.1 Requisitos funcionales

Los requisitos funcionales encontrados para el Sistema de Gestión de Licencias al Personal Aeronáutico del IACC son:

RF1. Insertar un usuario
RF2. Actualizar un usuario
RF3. Borrar un usuario
RF4. Listar los usuarios
RF5. Insertar un titular
RF6. Actualizar un titular
RF7. Borrar un titular
RF8. Listar los titulares
RF9. Insertar un titular externo
RF10. Actualizar un titular externo
RF11. Borrar un titular externo
RF12. Listar los titulares externo

RF13. Insertar un miembro de tripulación
RF14. Actualizar un miembro de tripulación
RF15. Borrar un miembro de tripulación
RF16. Listar los miembros de tripulación
RF17. Iniciar un tipo de licencia titular
RF18. Iniciar un tipo de licencia titular externo
RF19. Iniciar un tipo de licencia miembro tripulación
RF20. Adicionar habilitación a un titular
RF21. Adicionar habilitación a un titular externo
RF22. Renovar una licencia
RF23. Convalidar una licencia
RF24. Prorrogar una licencia
RF25. Generar notificaciones
RF26. Exportar Solicitud Jurada de iniciación de Licencia
RF27. Exportar Solicitud Jurada de renovación de Licencia
RF28. Exportar Solicitud Jurada de habilitación de Licencia
RF29. Exportar Solicitud Jurada de prórroga de Licencia
RF30. Exportar Solicitud Jurada de convocatoria de Licencia
RF31. Exportar una licencia
RF32. Exportar un certificado de validez
RF33. Exportar un certificado a miembro de tripulación
RF34. Insertar un Centro de Instrucción
RF35. Actualizar un Centro de Instrucción
RF36. Borrar un Centro de Instrucción
RF37. Listar los centros de instrucción
RF38. Insertar un programa de instrucción
RF39. Actualizar un programa de instrucción
RF40. Borrar un programa de instrucción
RF41. Listar los programas de instrucción
RF42. Insertar un simulador de vuelo
RF43. Actualizar un simulador de vuelo
RF44. Borrar un simulador de vuelo
RF45. Listar los simuladores de vuelo
RF46. Generar los siguientes reportes:

- Trámites realizados por meses y anual
- Estadística de la OACI

- Horas de vuelo de un tripulante técnico en un período de tiempo y por tipos de avión

2.4.2 Requisitos no funcionales

Para especificar los criterios que evalúan la operación del sistema, en contraste con los requisitos funcionales que especifican los comportamientos específicos se plantean los requisitos no funcionales.

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto usable, rápido o confiable. Los requerimientos no funcionales, aunque no aportan funcionalidades propiamente dichas dentro de la aplicación, son de vital importancia para una puesta en marcha exitosa del *software* y para lograr que este responda a las expectativas del usuario. (pmoinformatica, 2015)

Requisitos de *software* para estaciones clientes

- Navegador *web* Firefox versión 21 o superior
- Navegador Chrome versión 32 o superior

Requisitos de *software* para servidores *web*

- Servidor Apache 2,4 o superior
- PHP 5.3.9 o superior.
    1. Módulos PHP
    2. Libxml
    3. Curl
    4. Ctype
    5. Postgresql
    6. PDO
    7. PDOsql
    8. PDOsql_postgres
    9. Tokenixer
    10. Xml
    11. intl

Requisitos de *hardware* para servidores *web*

- PC Pentium 4 a 1 GHz o superior, mínimo 4GB de RAM, 40 GB o superior de disco duro

Requisitos de *software* para servidores de base de datos

- PostgresSQL 9.4 o superior.

Requisitos de *hardware* para servidores de base de datos

- PC Pentium 4 a 1 GHz o superior, mínimo 4 GB de RAM, 40 GB o superior de disco duro

Tecnologías que soportan la aplicación:

- Symfony 2.8

Requisitos no funcionales de usabilidad

- El sistema podrá ser utilizado por cualquier usuario con las siguientes características:
  - Conocimientos básicos relativos al uso de una computadora.
  - Conocimientos básicos del sistema operativo Windows.
  - Conocimientos sólidos relativos a los procesos de emisión de licencias aeronáuticas.
- Se brindará asesoramiento a los futuros usuarios del sistema en las funcionalidades que este ofrece.
- El sistema poseerá estructura y diseño homogéneos en todas sus pantallas, que facilite la navegación.

Requisitos no funcionales de seguridad.

- Se necesitará un usuario válido para acceder al sistema.
- El sistema constará con un usuario administrador.

2.5 Historias de usuario

Las historias de usuario son la técnica utilizada en AUP-UCI para especificar los requisitos del *software*. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer. Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto. La redacción de las mismas se hace bajo la terminología del cliente, de forma que sea sencilla, clara y no profundicen los detalles.

La prioridad en el negocio:

- **Alta:** cuando son consideradas por los clientes esenciales para el funcionamiento del negocio.
- **Media:** cuando el cliente cree que son necesarias, pero estas no intervienen en gran medida en el desarrollo del negocio.
- **Baja:** cuando constituyen procesos que se deben tener en cuenta, pero su ausencia no perjudica el flujo principal del negocio.

El riesgo en desarrollo:

- **Alto:** cuando en la implementación de las historias de usuario pueden surgir errores que lleven a la inoperatividad del código.
- **Medio:** cuando en la implementación de las historias de usuario pueden existir errores que retrasen la entrega del producto.
- **Bajo:** cuando pueden aparecer errores que serán tratados con relativa facilidad sin que traigan perjuicios para el desarrollo del proyecto.

A continuación, se muestra la historia de usuario de Adicionar Habilitación como ejemplo. (el resto de historias de usuarios en anexos)

RF20 Adicionar Habilitación

Número: RF04	**Nombre del requisito:** Adicionar habilitación
Programador:	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:**
Riesgo en Desarrollo: Medio	**Tiempo Real:**

Descripción:

El sistema permitirá en primera instancia listar tipo de licencia.

- Tipo de licencia (Cadena de Caracteres)
    1. Alumno Piloto (AP)
    2. Piloto privado (PP)
    3. Piloto comercial (PC)
    4. Piloto con tripulación múltiple (MPL) – avión
    5. Piloto de Transporte de Línea Aérea (PTLA)
    6. Licencia de Mecánico de a Bordo
    7. Licencia de Navegante
    8. Licencia de Auxiliar de a Bordo
    9. Licencia de Controlador de Tránsito Aéreo

10. Encargado de Operaciones de Vuelo
11. Técnico de Mantenimiento de Aeronaves
12. Operador de Estación Aeronáutica
13. AFIS

El sistema permitirá en primera instancia listar todos tipo de licencia.

El sistema mostrará todo el personal que tenga asociado el tipo de licencia asociado.

El sistema permitirá seleccionar el personal al cual añadir una nueva habilitación.

El sistema permitirá seleccionar un tipo y clase en dependencia del tipo de licencia.

Clase (monomotor terrestre, multimotor terrestre, aeronave de ala fija, …)

Tipo (aeromoza, capitán, copiloto, capitán de entrenamiento, …)

Observaciones:

El campo tipo depende del tipo de licencia. Esto está archivado en la Base de Datos de acceso que nos van a proporcionar.

Descripción para las siguientes licencias son aeropuertos:

1. Licencia de Controlador de Tránsito Aéreo
2. Encargado de Operaciones de Vuelo
3. Operador de Estación Aeronáutica

La descripción para las siguientes licencias son aviones:

1. Alumno Piloto (AP)
2. Piloto privado (PP)
3. Piloto comercial (PC)
4. Piloto con tripulación múltiple (MPL) – avión
5. Piloto de Transporte de Línea Aérea (PTLA)
6. Licencia de Mecánico de a Bordo
7. Licencia de Navegante
8. Licencia de Auxiliar de a Bordo
9. Técnico de Mantenimiento de Aeronaves

Prototipo de interfaz:

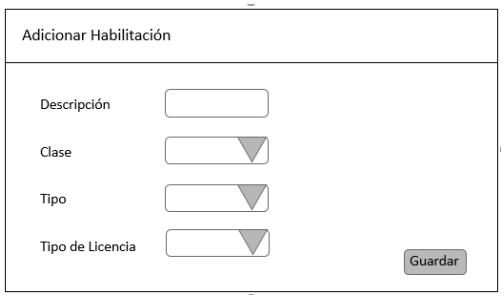


Imagen 3 Historia de Usuario(Adicionar habilitación)

2.6 Estilo Arquitectónico

Los estilos expresan la arquitectura en el sentido más formal y teórico, describen entonces una clase de arquitectura o piezas identificables de las arquitecturas empíricamente dadas. Esas piezas se encuentran repetidamente en la práctica, trasuntando la existencia de decisiones estructurales coherentes. Una vez que se han identificado los estilos, es lógico y natural pensar en reutilizarlos en situaciones semejantes que se presenten en el futuro. (Quiñones,2015)

Para el desarrollo del sistema se decide emplear el patrón arquitectónico Modelo - Vista - Controlador (MVC) ya que el marco de trabajo *Symfony2* brinda una estructura y un funcionamiento basado en la misma.

El patrón MVC se encuentra dentro de la clasificación de los estilos de llamada y retorno y tiene como objetivo separar la lógica de negocios de la interfaz gráfica de manera que cambios en la misma no afecten la lógica de negocios y viceversa. Está compuesto por tres componentes, el modelo que representa la información que tanto el usuario como la aplicación puede manipular, la vista que implica todos los elementos que componen la interfaz gráfica y el controlador que maneja la interacción y la comunicación entre el modelo y las acciones del usuario. (Álvarez, 2014)

Algunas ventajas de este patrón arquitectónico son:

- Facilita la agregación de múltiples representaciones de los mismos datos.
- Crea independencia de funcionamiento.
- Facilita el mantenimiento de errores.
- Alta escalabilidad pues se puede manejar muchas peticiones con el mismo rendimiento simplemente añadiendo más *hardware*.

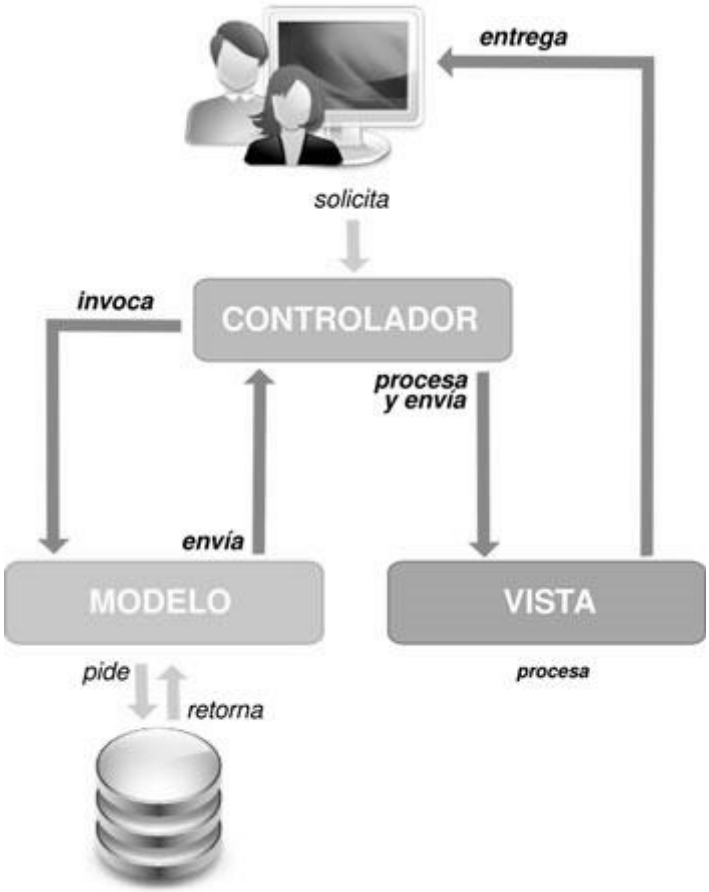


Imagen 4 Arquitectura MVC

Como se observa en la **imagen 3** a través del controlador son recibidas y atendidas todas las peticiones al sistema. Cuando el controlador recibe una petición del usuario, consulta los datos almacenados en el servidor *web* a través de un servicio desarrollado en *Symfony*.

2.7 Arquitectura

El concepto de arquitectura de *software* se refiere a la estructuración del sistema que, idealmente, se crea en etapas tempranas del desarrollo. Esta estructuración representa un diseño de alto nivel del sistema que tiene dos propósitos primarios: satisfacer los atributos de calidad (desempeño, seguridad, modificabilidad) y servir como guía en el desarrollo. Al igual que en la ingeniería civil, las decisiones críticas relativas al diseño general de un sistema de *software* complejo deben hacerse desde un principio. El no crear este diseño desde etapas tempranas del desarrollo puede limitar severamente el que el producto final satisfaga las necesidades de los clientes. Además, el costo de las correcciones relacionadas con problemas en la arquitectura es muy elevado. Es así que la arquitectura de *software* juega un papel fundamental dentro del desarrollo. (Cervantes, 2015) Es resumen, la arquitectura de *software* son las estructuras de un

sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos. (L. Bass, 2003)

En este caso es seleccionada la arquitectura Cliente-Servidor, esta es basada en el patrón arquitectónico MVC, donde la función del modelo la tienen los servicios de *Symfony* los cuales son los encargados de implementar la lógica del negocio gestionando todos los accesos a la información almacenada en servidor *web*.

2.8 Diagrama de clases del diseño

Los diagramas de clase pueden usarse cuando se desarrolla un modelo de sistema orientado a objetos para mostrar las clases en un sistema y las asociaciones entre dichas clases (Sommerville, 2012). Los diagramas de clases del diseño con estereotipos web se utilizan para describir gráficamente las especificaciones del modelo, la vista y el controlador. Estas representaciones contienen información acerca de las clases, asociaciones, atributos, métodos y dependencias.

A continuación, se muestra el diagrama de clases del diseño para el requisito funcional adicionar habilitación.

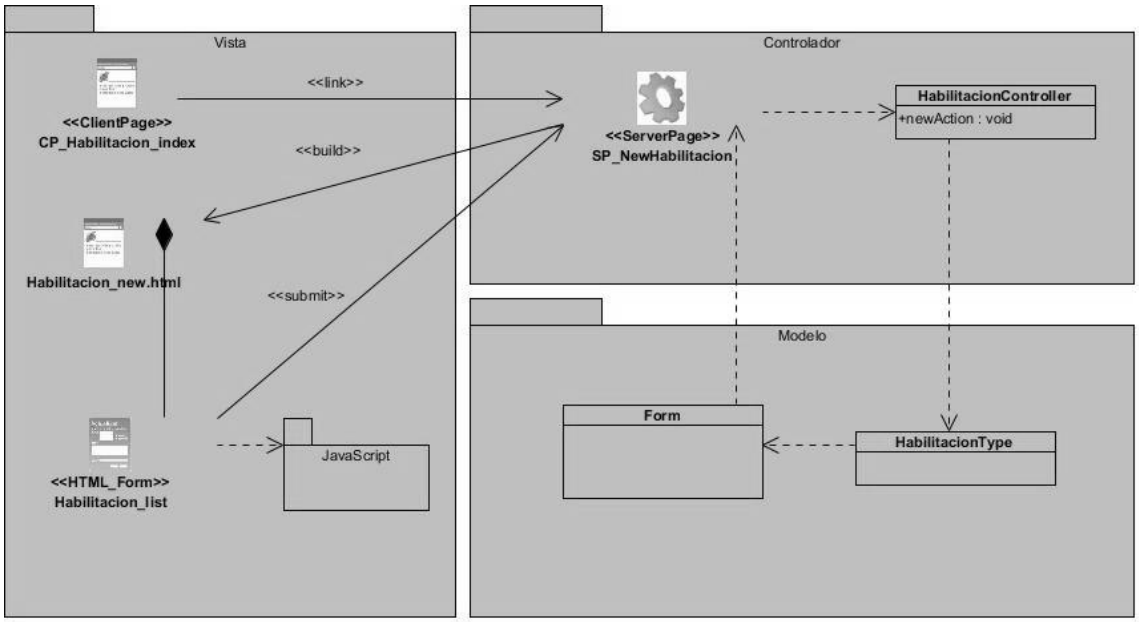


Imagen 5 Diagrama de clases del diseño del RF adicionar habilitación

2.9 Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de *software* y otros ámbitos referentes al diseño de interacción o interfaces. Son parejas de problema/solución con un nombre, que codifican buenos

principios y sugerencias relacionados generalmente con la asignación de responsabilidades. (Agudelo, A, 2015)

En el diseño de la herramienta se tuvieron en cuenta los patrones GRASP (Patrones Generales de *Software* para Asignación de Responsabilidades), los cuales describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.

De los patrones GRASP se utilizan los siguientes:

Experto: este patrón plantea que se debe asignar una responsabilidad al experto en información, o sea, a la clase que cuenta con la información necesaria para cumplir la responsabilidad.

Beneficios del patrón Experto:

- Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto posibilita tener sistemas de fácil mantenimiento.
- El comportamiento se distribuye entre las clases que cuentan con la información requerida para cumplir con la responsabilidad asignada.

En la solución implementada se hace uso de este patrón en la clase TipoLicenciaController, debido a que esta clase conoce la información suya propia y la de Habilitacion, por lo cual puede hacer modificaciones y llamadas a ésta, para reutilizar código.

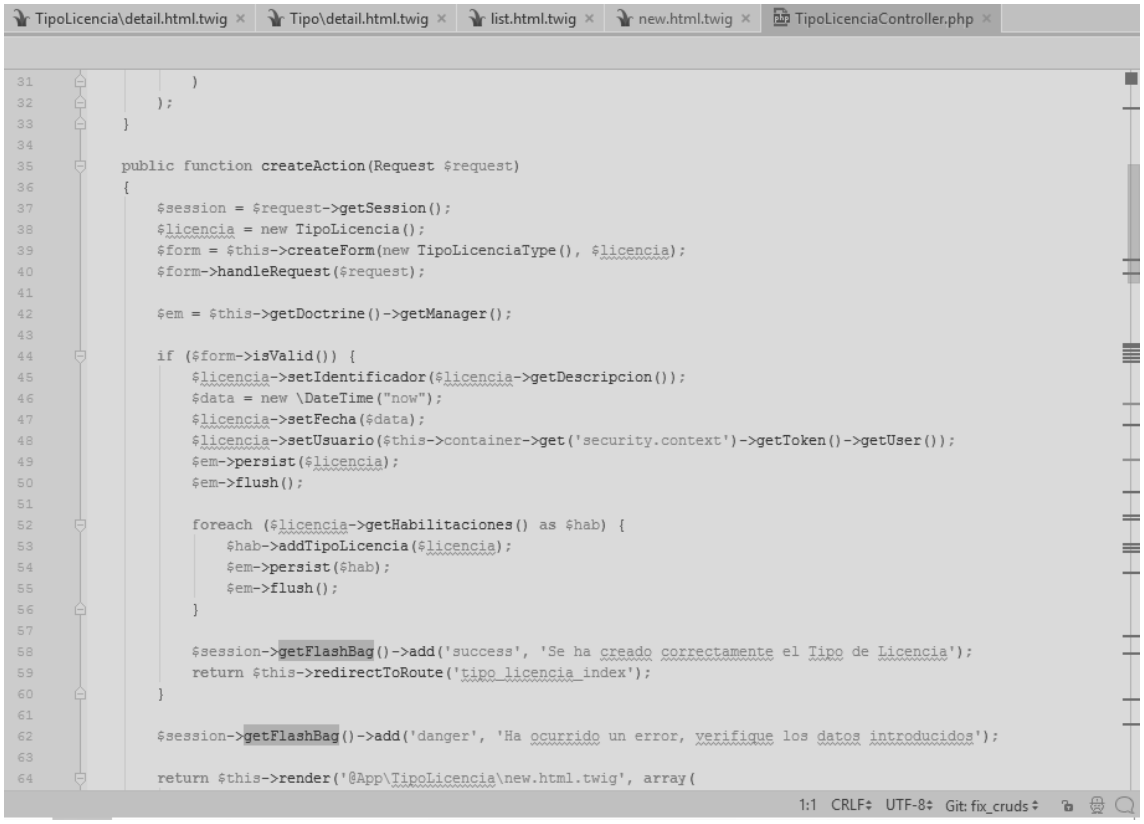


Imagen 6 Patrón de diseño Experto de la clase TipoLicenciaController

Alta cohesión: este patrón plantea que se debe asignar una responsabilidad de modo que la cohesión siga siendo alta (una clase tiene responsabilidades moderadas). Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas, que no realicen un trabajo enorme. Una clase con baja cohesión hace muchas cosas no afines o un trabajo excesivo.

Beneficios del patrón Alta cohesión

- Mejoran la claridad y la facilidad con que se entiende el diseño.
- Se simplifican el mantenimiento y las mejoras en funcionalidad.

Una de las características de Symfony es la organización del trabajo en cuanto a la estructura del proyecto, lo cual permite crear y trabajar con clases con una alta cohesión. Por ejemplo, se puede observar en el sistema que cada clase controladora se ajusta a manejar solo las responsabilidades correspondientes a las entidades con las que se relaciona. Esto hace posible que el sistema sea flexible a cambios sustanciales con efecto mínimo.

Controlador: sirve como intermediario entre una interfaz y el algoritmo que la implementa, es el que recibe los datos del usuario y el que los envía a las distintas clases de acuerdo al método llamado. Sugiere que la lógica de negocios debe estar

separada de la presentación, así se logra un mayor control, además de la reutilización de código.

Este patrón es empleado en todo el sistema debido a que cada uno de los eventos generados por el usuario es redirigido a una clase o función controladora que realiza las operaciones solicitadas, manteniendo siempre la alta cohesión. A continuación, se muestra la clase TipoLicenciaController, una de las clases controladoras y sus funcionalidades.

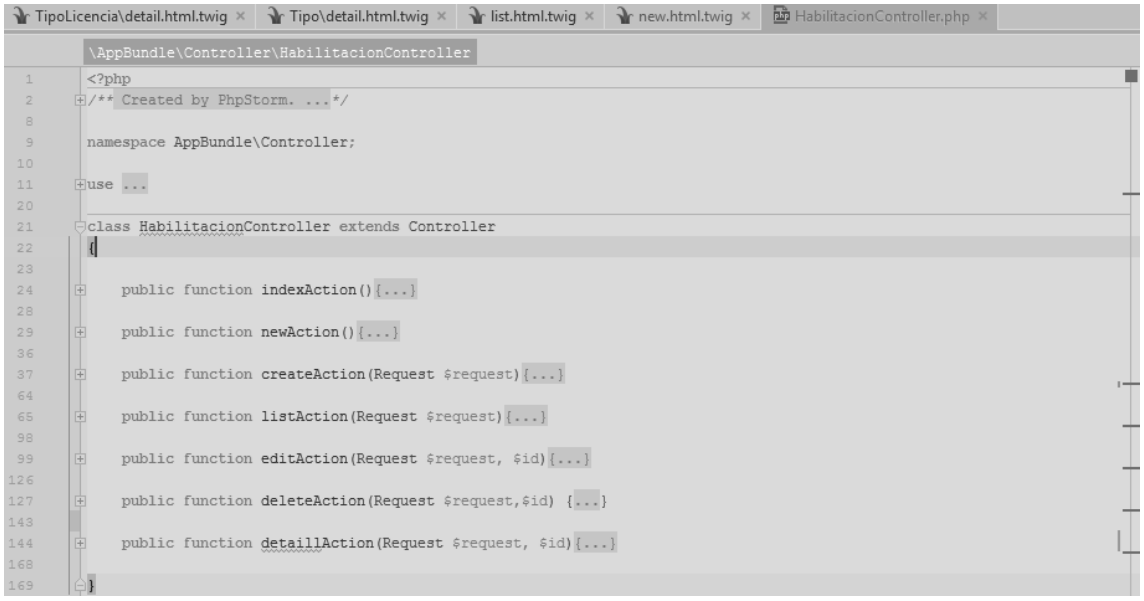


Ilustración 7 Patrón de diseño controlador de la clase HabilitacionController

2.10 Modelo de Datos

A continuación, se muestran los diagramas de clases del diseño entidad-relación en el que se muestran las tablas de la base de datos, así como sus atributos y relaciones mediante un modelo UML.

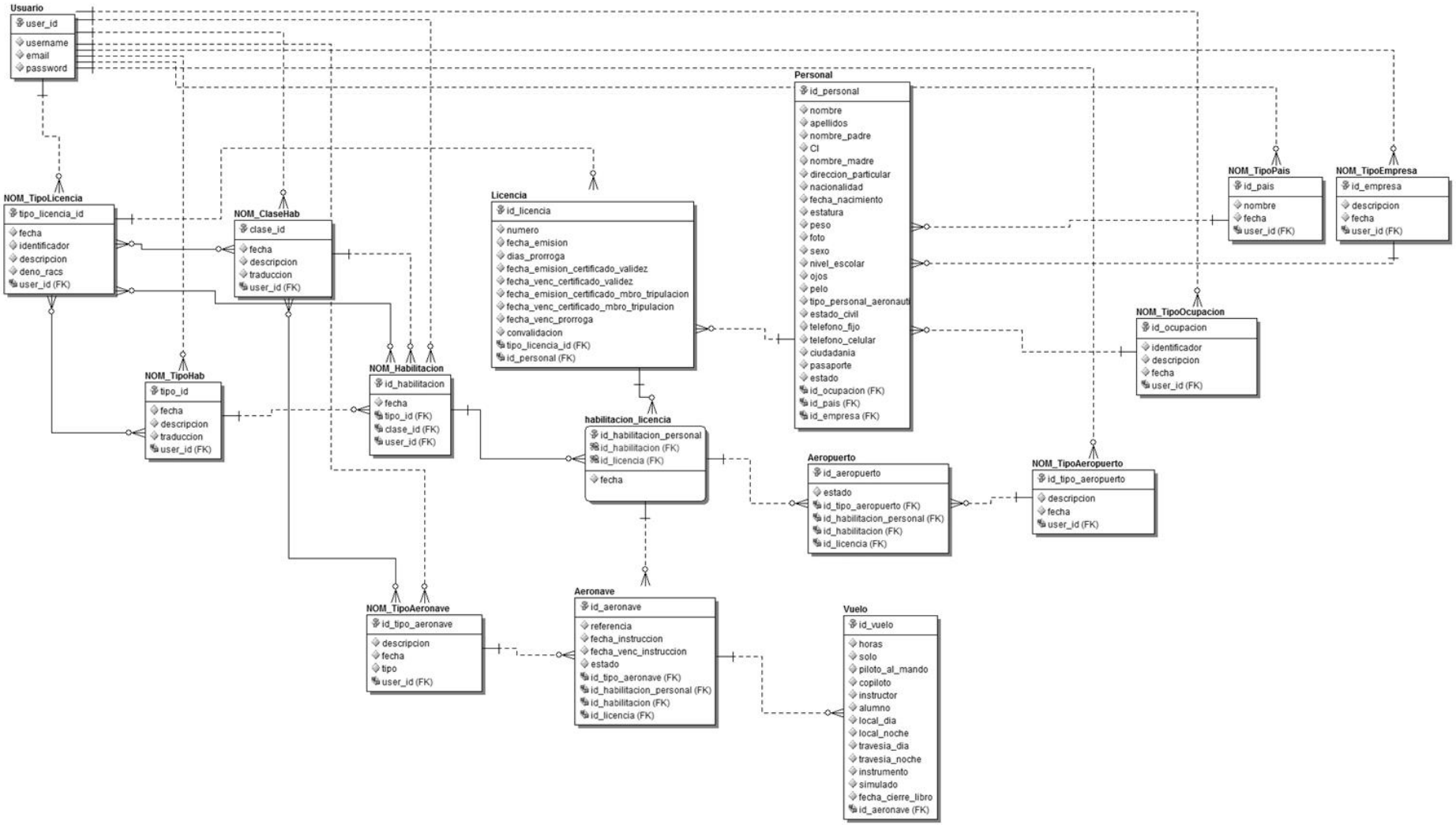


Imagen 8 Modelo de datos

2.11 Modelo de despliegue

Para mejor comprensión entre la correspondencia de la arquitectura de *software* y la arquitectura de *hardware* se realiza el modelo de despliegue.

El modelo de despliegue que se presenta a continuación muestra el nodo "Dispositivo_Cliente", representado por un nodo ordenador el cual contiene un navegador para Internet. Este se encarga de comunicarse con el nodo que contiene la aplicación *web* a través del protocolo HTTP. Este proceso se realiza a través de los recursos que se muestran al usuario en la página, lo que le permite establecer un sistema de comunicación con el servidor *web* Apache.

En el nodo "Servidor *web*" se atienden las solicitudes del cliente, se analizan y se les da respuesta. En este nodo están contenidos los procesos de información que garantizan el funcionamiento de la aplicación logrando cumplir con los requerimientos funcionales del sistema. La capa del modelo se comunica con el nodo "Servidor de Bases de Datos Apache" a través del protocolo TCP, donde se encuentra la información almacenada.

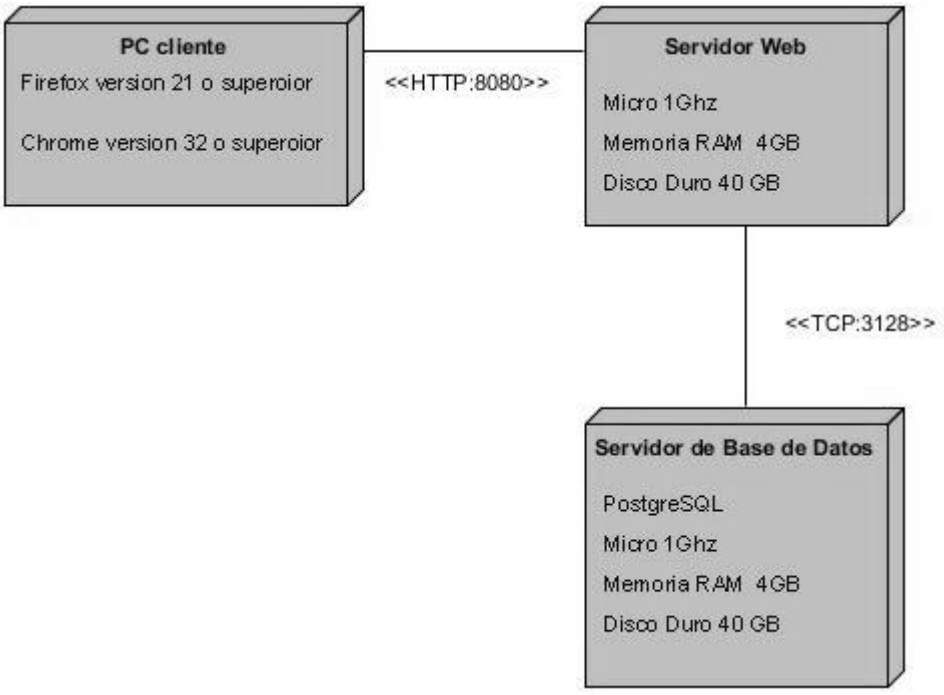


Imagen 9 Modelo de Despliegue

2.12 Conclusiones parciales

Con la especificación de los requisitos funcionales y no funcionales del sistema, se logró un mejor entendimiento, de los resultados que se pretenden obtener y sirvieron de guía para la implementación del sistema. La representación y descripción de los artefactos generados garantizaron una mejor comprensión de los flujos de trabajos presentes. La definición de la arquitectura y los patrones de diseño a utilizar, permitieron establecer

las bases para fomentar la reutilización y las buenas prácticas de programación durante la fase de implementación, así como disminuir el impacto de los cambios futuros en el código. La elaboración del diagrama de despliegue permitió identificar la disposición física de los artefactos del producto informático a desarrollarse.

Capítulo 3. Implementación y prueba.

3.1 Introducción

La fase de implementación en el desarrollo de un producto de *software*, es el mecanismo donde se ponen en práctica todas las descripciones y arquitecturas propuestas en las fases de análisis y diseño, es el complemento del trabajo de las fases que lo preceden dentro del proceso de desarrollo de *software*. La implementación ofrece una materialización precisa de los requisitos.

Una de las últimas fases del ciclo de vida antes de entregar un programa para su explotación es la fase de pruebas, cuyo objetivo es comprobar si este cumple sus requisitos. Dentro de ella pueden desarrollarse varios tipos de pruebas en función de los objetivos de las mismas.

3.2 Modelo de componentes que integran la solución informática

El modelo de componentes representa la forma en que es estructurado un sistema informático atendiendo a las diferentes partes que lo componen. Partiendo de este punto (Sommerville, 2012), puntualiza que cada componente debe ser tratado como una unidad de composición independiente e indispensable dentro de un sistema, y que puede contraer relaciones de dependencia con otros componentes. Algunos ejemplos de componentes físicos lo constituyen los archivos, módulos, librerías, ejecutables, binarios, entre otros.

3.3 Diagrama de componentes

El diagrama de componentes describe cómo se implementan las clases en término de componentes, como pueden ser, ficheros de código, ejecutables, entre otros. Describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponible en el entorno de implementación y en el lenguaje de programación utilizado. Además, muestra las dependencias entre componentes. (Toro, 2016).

A continuación, se muestra el diagrama de componentes propuesto, el cual está acorde con los requerimientos del sistema a implementar y con el patrón basado en componentes.

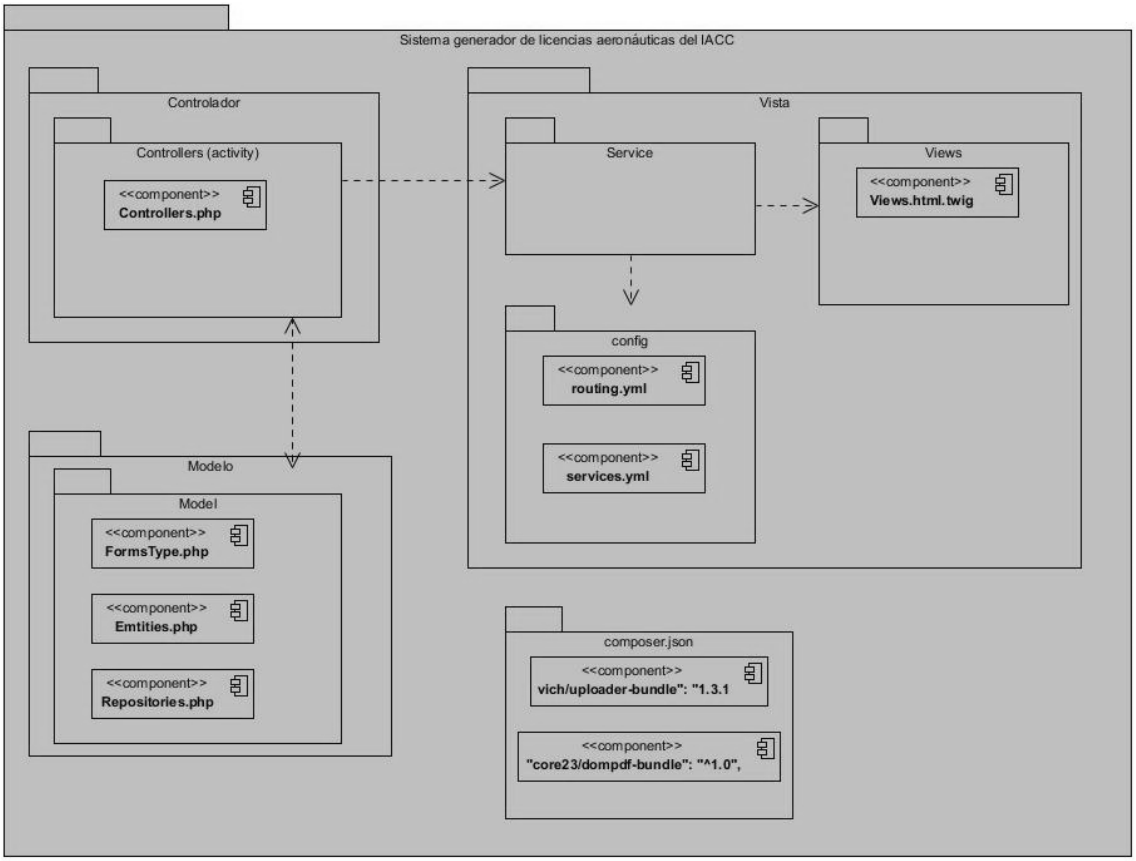


Imagen 10 Diagrama de componentes

3.4 Estándar de codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez, para facilitar su entendimiento, fijar un modelo a seguir y establecer un estándar de codificación. Debido a la utilización del *framework Symfony2* la implementación conlleva el obligatorio estándar de codificación establecido por el mismo para el funcionamiento del sistema.

Estructura

- Añade un solo espacio después de cada delimitador coma.
- Añade un solo espacio alrededor de los operadores (==, &&, ...).
- Añade una coma después de cada elemento del arreglo en un arreglo multilínea, incluso después del último.

- Añade una línea en blanco antes de las declaraciones *return*, a menos que el valor devuelto solo sea dentro de un grupo de declaraciones (tal como una declaración if).
- Usa llaves para indicar la estructura del cuerpo de control, independientemente del número de declaraciones que contenga.
- Define una clase por archivo — esto no se aplica a las clases ayudante privadas, de las cuales no se tiene la intención de crear una instancia desde el exterior.
- Declara las propiedades de clase antes que los métodos.
- Declara primero los métodos públicos, luego los protegidos y finalmente los privados.
- Utiliza paréntesis al instanciar clases independientemente del número de argumentos que tenga el constructor.
- Las cadenas de mensajes de excepción deben concatenarse mediante *sprintf*.

Convenciones de nomenclatura

- Utiliza mayúsculas intercaladas (sin guiones bajos) en nombres de variable, función, método o argumentos.
- Usa guiones bajos para nombres de opción y nombres de parámetro.
- Utiliza espacios de nombres para todas las clases.
- Prefija las clases abstractas con *Abstract*.
- Sufija las interfaces con Interface.
- Sufija las características con *Trait*.
- Sufija las excepciones con *Exception*.
- Utiliza caracteres alfanuméricos y guiones bajos para los nombres de archivo.

Documentación

- Añade bloques *PHPDoc* a todas las clases, métodos y funciones.
- Omite la etiqueta @return si el método no devuelve nada.
- Las anotaciones *@package* y *@subpackage* no se utilizan.

(Symfony, 2013)

3.4.1 Nomenclatura según el tipo de clases

Debido a la utilización del *framework Symfony2* es usado este estándar de codificación en todo el proyecto y se puede ver en diferentes clases del mismo como en:

Clases controladoras: las clases que se encuentran dentro de la carpeta controller después del nombre de la clase incorporan la palabra: “Controller”, ejemplo: MbroTripulaciónController, TipoAvionController.

Formularios: las clases que se encuentran dentro de la carpeta Form después del nombre de la clase incorporan la palabra “Type”, ejemplo: TipoPaisType, HabilitacionType.

3.4.2 Nomenclatura de las funcionalidades y atributos

El nombre a emplear para las funciones y los atributos se escriben con la inicial minúscula, en caso de que sea un nombre compuesto se utiliza una mayúscula para definir dónde empieza la segunda palabra y es necesario agregar la palabra Action al final del nombre del método para la funcionalidad de *Symfony,* ejemplo: createAction(), listAction(),

3.5 Interfaces de Usuario

El sistema desarrollado debe poseer la interfaz establecida por la universidad para este proyecto. Dicho sistema cuenta con dos interfaces principales; la primera utilizada para el *login* y la segunda para todas las otras *view* del proyecto. Las imágenes 7 y 8 hacen referencia a las interfaces antes mencionadas.

Imagen 11 Login

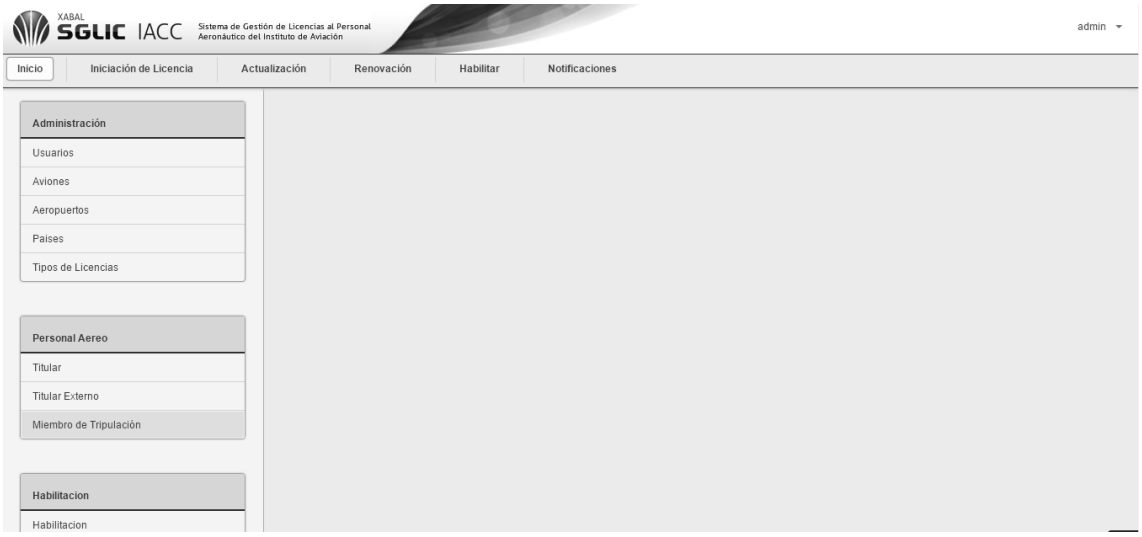


Imagen 12 Interfaz de usuario

3.6 Pruebas y resultados

A continuación, se detallan los tipos de pruebas de *software* aplicados al sistema. Las mismas persiguen como objetivo fundamental, la detección de las no conformidades

respecto a las funcionalidades de la aplicación, las vulnerabilidades que atentan contra la seguridad de la información que se manipula con el *software*, la medición del grado de usabilidad de las funcionalidades implementadas, así como también la correcta integración entre los diferentes componentes de la arquitectura del sistema.

3.6.1 Pruebas unitarias

Las pruebas unitarias de Symfony son archivos PHP normales cuyo nombre termina en Test.php y se encuentran en el directorio AppBundle/test/ de la aplicación. Su sintaxis es sencilla y fácil de leer.

Las pruebas unitarias que se crean deben implementarse con el uso de una estructura que permita automatizarlas, de modo que puedan ejecutarse en reiteradas ocasiones y con facilidad. Esto estimula una estrategia de pruebas de regresión, siempre que se modifique el código. El programador es el encargado de escribir las pruebas unitarias y producir el código del sistema. Son aplicables a funcionalidades para verificar que los flujos de control y de datos están cubiertos y funcionando como se espera. (Symfony, 2017)

Resultados de pruebas funcionales

Para la realización de estas pruebas se utilizó la *librería* de *PHP* llamada PHPUnit_Framework_TestCase que permite detectar errores en el funcionamiento de determinadas *clases, métodos* o *directorios*. Se aplicó el código de prueba sobre las diferentes funcionalidades del sistema una vez fueron terminadas ayudando a comprobar si las mismas mostraban el resultado esperado.

A continuación, se pueden observar los resultados de las pruebas unitarias aplicadas a las clases MbroTripulacionController y NomencladorController.

La prueba unitaria sobre la clase controladora MbroTripulacionController para cada uno de sus *métodos* confirmó que ésta realiza sus funciones correctamente (**Imagen 9**).

Imagen 13 Resultados de prueba unitaria sobre MbroTripulacionController Iteración 1

Al aplicarse la prueba a la clase NomencladorController en una primera ocasión ésta mostró que dos de sus funcionalidades no devolvían el valor esperado, lo que conllevó

a la rectificación del código y una segunda iteración para comprobar si se corrigieron los errores (**imagen 10 y 11**).

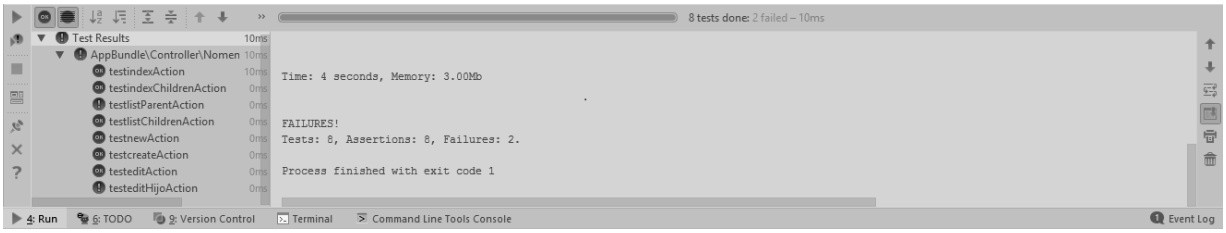


Imagen 14 Resultados de prueba unitaria sobre NomencladorController Iteración 1

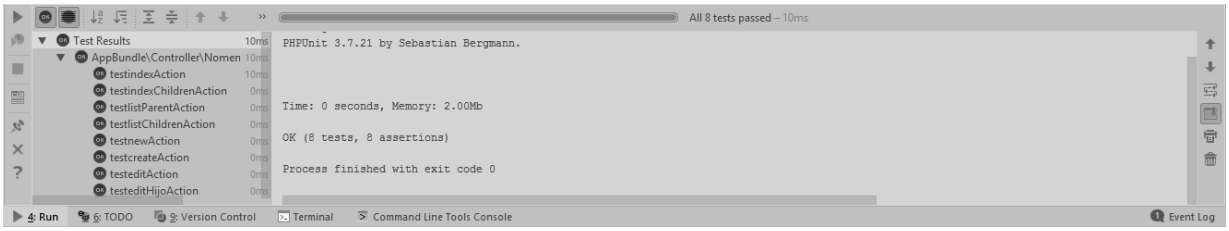


Imagen 15 Resultados de prueba unitaria sobre NomencladorController Iteración 2

3.6.2 Pruebas funcionales

Las pruebas funcionales son aquellas que se llevan a cabo sobre la interfaz del *software* sin prestar atención al código, por lo que los casos de prueba son creados con el objetivo de demostrar que la entrada es aceptada de forma adecuada y que se produce una salida correcta. El diseño de estas pruebas se realiza con la intención de detectar funciones incorrectas o ausentes, errores en accesos a bases de datos externas, errores de interfaz, errores de rendimiento y errores de inicialización y de terminación. Dentro de la prueba se incluyen la técnica de partición de equivalencia que será la empleada en la validación. Esta técnica divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba (Pressman, 2010).

Se realizaron los casos de prueba para validar el funcionamiento de la propuesta de solución implementada para comprobar que el sistema agregue correctamente un tipo de licencia. A continuación, en las **Tablas 3 y 4**, se muestra la especificación del caso de prueba para el requisito “Crear tipo de licencia”.

No.	Nombre del campo	Clasificación	Valor Nulo	Descripción
1	Metatipo	Enum	No	Puede ser PIL, MBRO o PERS
2	Identificador	Área de Texto	No	Contiene todo tipo de

				caracteres
3	Descripción	Área de Texto	No	Contiene todo tipo de caracteres
3	Habilitaciones	Campo de selección	No	Contiene las habilitaciones

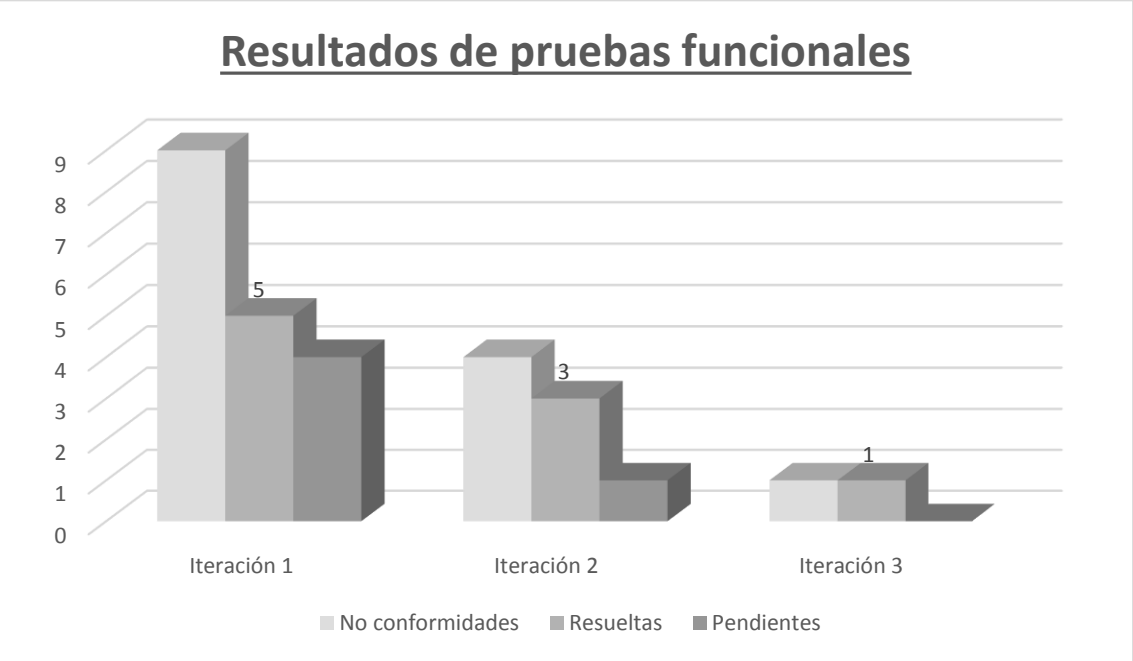
Tabla 3 Variables del caso de prueba para el escenario Adicionar Tipo de Licencia

Escenario	Descripción	1	2	3	5	Respuesta	Flujo central
Ec. 1 Crear tipo de licencia. Datos correctos	Se registran los datos del nuevo tipo de licencia.	Se selecciona PIL, MBRO o PERS	Nombre identificativo del tipo de licencia	Descripción de la licencia	Se selecciona una habilitación	Muestra el mensaje “Se ha creado correctamente el Tipo de Licencia*”*	El operario introduce los datos de forma correcta y da clic en el botón Guardar.
Ec. 2 Crear tipo de licencia. Datos correctos	Existen campos obligatorios vacíos	Se selecciona PIL, MBRO o PERS	Nombre identificativo del tipo de licencia	(Vacío)	No se selecciona una habilitación	Muestra el mensaje *“Ha ocurrido un error, verifique los datos introducidos)”*	El operario introduce los datos pero deja alguno campos vacíos y da clic en Guardar.

Tabla 4 Casos de prueba Introducir Tipo de Licencia

Resultados de pruebas funcionales

Las pruebas funcionales se realizaron en tres iteraciones a medida que avanzaba el sistema, las cuales guiaron la calidad del mismo y determinaron en cada momento si se estaba o no en condiciones de continuar avanzando con el ciclo de desarrollo. Estas pruebas arrojaron en un principio un total de nueve no conformidades, siendo principalmente errores ortográficos, en la lógica del funcionamiento y validaciones incorrectas; logrando dar solución a 5 de ellas. En una segunda iteración se detectaron cuatro no conformidades las cuales fueron en cuanto a errores lógicos y de validaciones, las cuales fueron corregidas. Por último, se realizó la tercera iteración detectando solo una no conformidad de validación la cual fue inmediatamente corregida.



Al concluir estas pruebas tras el análisis de los resultados se puede ver que los resultados de las mismas fueron satisfactorios. Pudieron ser detectadas las no conformidades, a las cuales se les dio solución, logrando así el correcto funcionamiento del sistema ante diferentes situaciones.

3.6.3 Pruebas de Seguridad

Las pruebas de seguridad buscan medir la Confidencialidad, Integridad y Disponibilidad de los datos, desde la perspectiva del aplicativo, es decir partiendo de identificar amenazas y riesgos desde el uso o interface de usuario final. Una vez ejecutadas las pruebas de seguridad es posible medir y cuantificar los riesgos a los cuales se ven expuestos los aplicativos tanto en la infraestructura interna como externa, valiéndose de la filosofía del Hacking ético. (vyvquality, 2016)

Para la realización de las pruebas de seguridad se usó la herramienta de detección de vulnerabilidades *Acunetix Web Vulnerability Scanner 9.5*, la cual arrojó un total de 12 vulnerabilidades, que se clasifican en alta, baja, media e informativas (**Imagen 9**).

En la primera iteración no se detectó ninguna vulnerabilidad de alta complejidad; de complejidad media se detectaron tres vulnerabilidades. La primera, que las credenciales de usuario se transmiten a través de un canal sin cifrar; ésta información siempre debe transferirse a través de un canal cifrado (HTTPS) para evitar ser interceptada por usuarios malintencionados. Se detectó también la vulnerabilidad "*Slow HTTP* Denial of Service attacks", se basan en el hecho de que el protocolo HTTP, por diseño, requiere que las solicitudes sean completamente recibidas por el servidor antes de que se procesen. Si una solicitud HTTP no está completa o si la velocidad de transferencia es muy baja, el servidor mantiene sus recursos ocupados esperando el resto de los datos. Si el servidor mantiene demasiados recursos ocupados, esto crea una denegación de servicio. En el caso del sistema generador de licencias no constituye una amenaza relevante pues éste no está pensado para una alta cantidad de usuarios. Otra vulnerabilidad de nivel medio detectada consiste en que las credenciales del usuario son enviadas por un canal sin encriptar.

De baja complejidad se detectaron 6 vulnerabilidades, entre ellas se encuentran la existencia de posibles *virtual host*, *Cookies* de sesión sin conjunto de indicadores seguros y el método HTTP TRACE está habilitado en este servidor *web*. Esto permite que los atacantes pueden abusar de la funcionalidad HTTP TRACE para obtener acceso a información en encabezados HTTP como cookies y datos de autenticación.

Se encontró también una vulnerabilidad de nivel informativo referente a las contraseñas de entrada que están guardadas en el navegador y usan el auto completamiento del mismo.

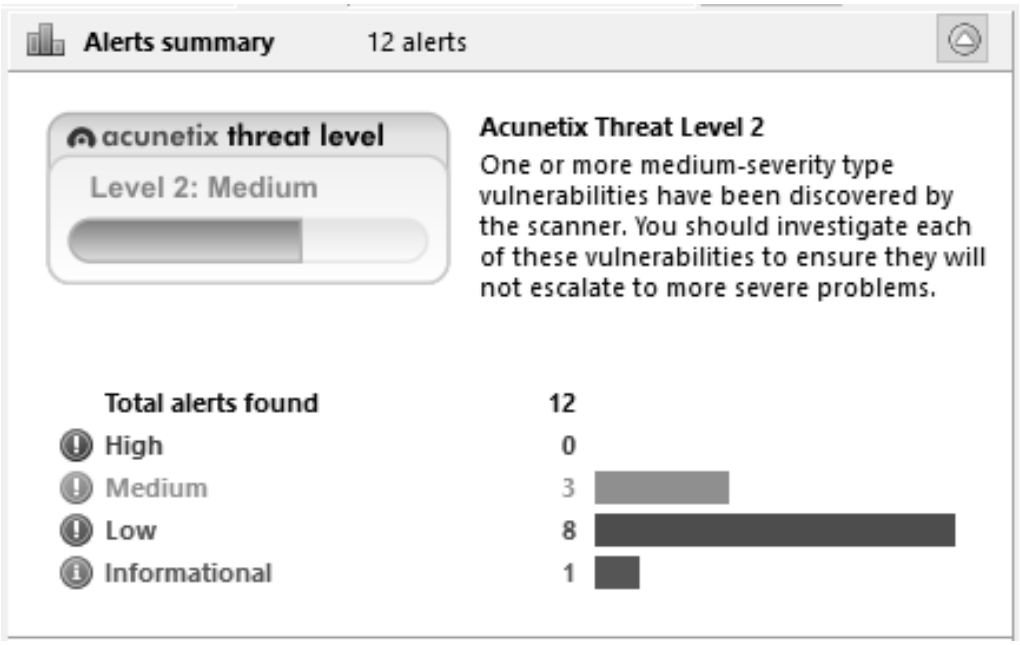


Imagen 16 Prueba de seguridad iteración 1 (Acunetix)

Para la segunda iteración se corrigieron 10 vulnerabilidades (**imagen 10**) de las detectadas en la iteración 1, mientras que en esta iteración se detectaron dos alertas una de nivel medio y otra de carácter informativo donde se mantuvo la vulnerabilidad *"*Slow HTTP *Denial of Service attacks"* y la vulnerabilidad de nivel informativo de la contraseña guardada en el navegador.

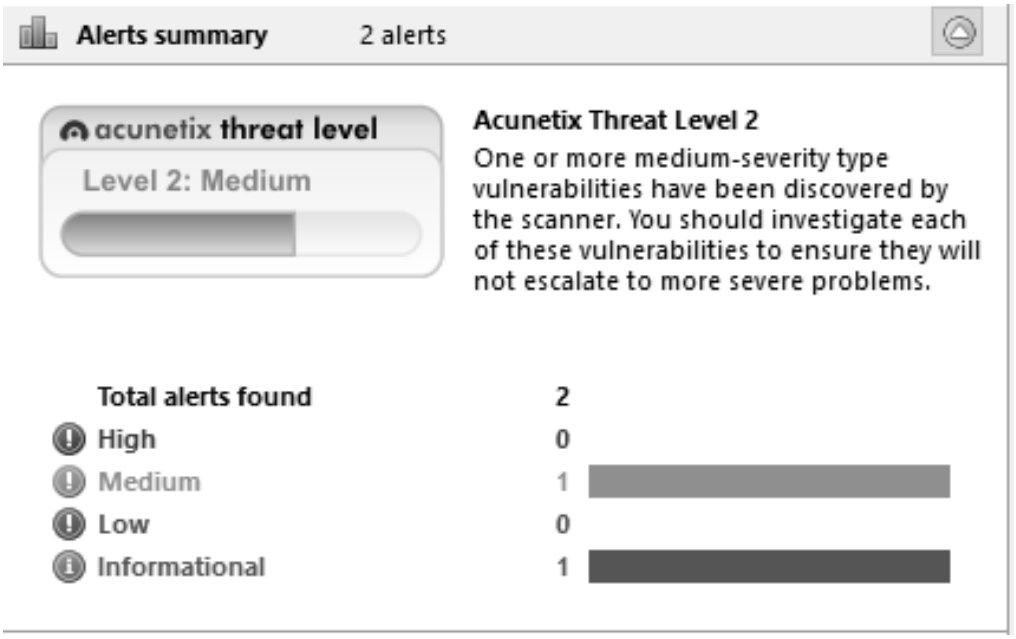


Imagen 17 Prueba de seguridad iteración 2 (Acunetix)

Finalmente, para la tercera iteración se logró corregir todas las brechas de seguridad logrando una integración y confiabilidad de los datos (**imagen 11**).

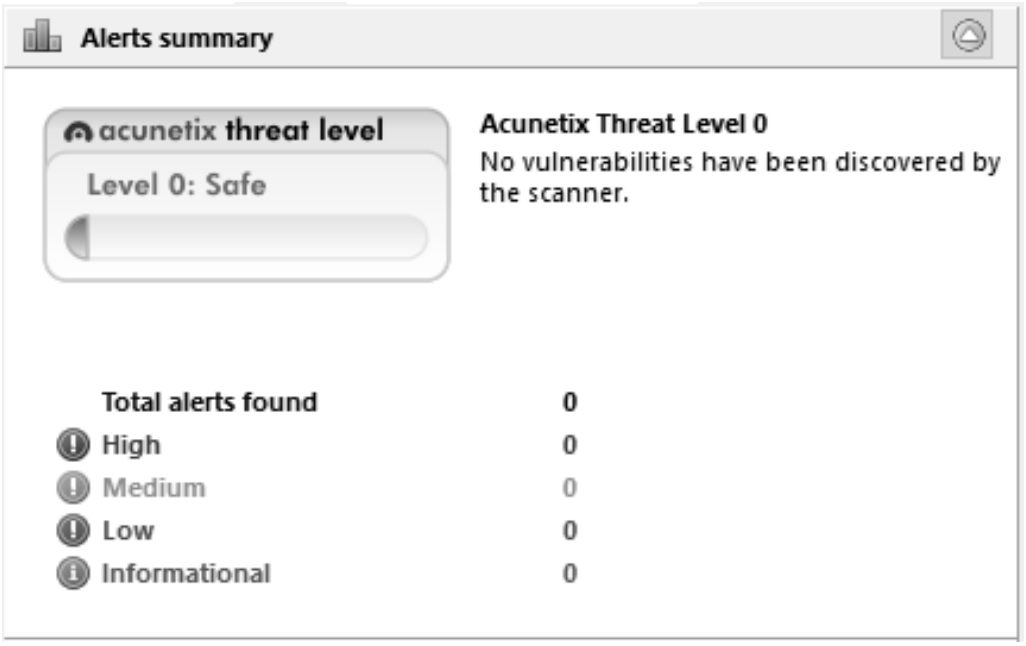


Imagen 18 Prueba de seguridad iteración 3 (Acunetix)

3.7 Conclusiones parciales

En este capítulo se abordaron una serie de aspectos correspondientes a la implementación y validación del sistema desarrollado. La representación y descripción del diagrama de componentes permitió visualizar con más facilidad la estructura general de la herramienta. La ejecución de pruebas a la herramienta permitió detectar las deficiencias presentes, subsanarlas en el menor tiempo posible y ofrecer una aplicación con mayor calidad, seguridad y usabilidad. Como resultado de esta fase se obtiene un producto funcional para entregar al cliente, el que debe garantizar el cumplimiento de los requisitos estipulados con la calidad requerida.

Conclusiones

A raíz de la investigación que se realizó sobre los sistemas de gestión de licencias a nivel internacional y nacional funcionales actualmente, la selección de la metodología AUP-UCI y del marco de trabajo a utilizar en el desarrollo, se confeccionaron las bases para implementar el Sistema Generador de Licencias para el Instituto de Aeronáutica Civil de Cuba.

El correcto análisis y diseño de la propuesta de solución permitió una mayor comprensión sobre el dominio del problema de la investigación. Se definió la arquitectura Cliente-Servidor y el patrón arquitectónico Modelo-Vista-Controlador. Las historias de usuario, permitieron elaborar de manera correcta los requisitos funcionales y no funcionales del sistema en cuestión.

En la fase de implementación se utilizó el lenguaje *PHP* con el *framework Symfony* y se desarrolló de manera exitosa el Sistema Generador de Licencias, el cual da solución a los problemas actuales existentes en el Instituto de Aeronáutica Civil de Cuba, presentes en la oficina de gestión de licencias. Durante el desarrollo se puso en práctica los patrones y estándares de codificación planificados.

Se realizaron una serie de pruebas al producto final, las cuales permitieron detectar los errores presentes en la solución desarrollada y corregirlos en el menor tiempo posible. De esta forma se logra un producto final que garantice el cumplimiento de los requisitos estipulados con la calidad requerida, lo que permite comprobar que se ha cumplido con el objetivo propuesto para el presente trabajo debido a que se logra un mejoramiento en las condiciones de trabajo del cliente y la eficacia de los procesos ahí realizados.

Bibliografía

Agencia estatal de Seguridad Aérea: Ministerio de Fomento. 2016. AESA. [En línea] 2016. www.seguridadaerea.gob.es.

ALEGSA. 2016. Diccionario de Informática y Tecnología. [En línea] diciembre de 2016. http://www.alegsa.com.ar/Dic/python.php.

Álvarez, Miguel Angel. 2014. DesarrolloWeb.com. [En línea] 2014. https://desarrolloweb.com/articulos/que-es-mvc.html.

Avante. 2013. Avante. [En línea] 2013. http://www.avante.es/comparativa-metodologias-agiles-vs-tradicionales/.

Borja, Yolanda. 2016. http://www.runayupay.org/. *Metodología Ágil de Desarrollo de Software - XP.* [En línea] 2016. http://www.runayupay.org/publicaciones/2244_555_COD_18_290814203015.pdf.

Cardentey, Juan Antonio González. 2009. *Trabajo de Diploma para optar por el título de Ingeniero Informático: "Desarrollo de una aplicación generadora de Servicios Web".* 2009.

Cervantes, Humberto. 2015. SG Buzz. [En línea] 2015. https://sg.com.mx/revista/27/arquitectura-software.

De Nobrega, María. Herramientas CASE: Rational Rose. [En línea] [Citado el: 18 de noviembre de 2014.] http://curso_sin2.blogia.com/2005/060401-herramientas-case-rational-rose..

DIRECCIÓN GENERAL DE AERONÁUTICA CIVIL. 2016. SISTEMA DE LICENCIAS AERONÁUTICAS ALVI. [En línea] 2016. https://www.alvidgac.gob.cl/webapps/index.php/login/index.

Direccion General de Aeronáutica Cuvil, Ministerio de Comunicaciones, Infraestructura y vivienda. 2016. DGAC. [En línea] 2016. https://www.dgac.gob.cl/portalweb/dgac/.

Direccion General de Aviación Civil. 2015. Ecuador ama la vida. [En línea] 27 de abril de 2015. http://www.aviacioncivil.gob.ec/?p=4512.

Doutdeslibretas. 2011. Doutdeslibretas. [En línea] 2011.

EcuRed. 2017. EcuRed. [En línea] 20 de 1 de 2017. https://www.ecured.cu/Metodolog%C3%ADa_%C3%A1gil.

—. 2017. EcuRed. [En línea] enero de 2017. https://www.ecured.cu/Symfony.

Eguiluz, Javier. 2014. *Desarrollo web ágil con Synfony2.* 2014.

Enciclopedia jurídica. 2014. [En línea] 2014. http://www.enciclopedia-juridica.biz14.com/d/personal-aeronautico/personal-aeronautico.htm.

ESI. 2016. Escuela de Sistemas Informaticos. [En línea] diciembre de 2016. http://www.falconmarbella.com/esigranada/dmdocuments/APACHE.p.

Espino. 2017. [En línea] enero de 2017. http://www.v-espino.com/~chema/daw1/tutoriales/postgres/pgadmin1.htm.

Feitó Cabrera, Marta Elena, González López, Odalys y Portuondo Brizuela, Alexi. 2012. *Pautas para un cambio en el servicio de pago de las pensiones de seguridad social.* La Habana : s.n., 2012.

García, Rosana Castro. 2015. *Trabajo de diploma para optar por el titulo de Ingeniero: Módulo de control de asistencia mediante identificación por huellas dactilares.* 2015.

Guzmán, Pilar Arellano y zTyhM. 2011. Programación de pag.web. [En línea] 9 de mayo de 2011. http://progpagweb.blogspot.com/2011/05/definicion-y-caracteristicas-de-php-con.html.

IACC. 2017. Instituto de Aerónautica Civil de Cuba. [En línea] 10 de enero de 2017. http://www.iacc.gob.cu/.

Instituto Tecnológico de Toluca. 2009. Scribd. [En línea] 2009. https://es.scribd.com/document/21938147/CUADRO-COMPARATIVO.

integra. 2016. integra. Consultores de Sistemas de Gestión. [En línea] 2016. http://www.consultoresdesistemasdegestion.es/sistemas-de-gestion/.

Internet Ya. 2017. Internet Ya Soluciones Web. [En línea] 20 de enero de 2017. http://www.internetya.co/ventajas-y-beneficios-de-las-aplicaciones-web/.

L. Bass, P. Clements, R. Kazman. 2003. *Software Architecture in Practice, 2nd Edition.* s.l. : Addison Wesley, 2003.

Letelier, Patricio y Penadés, Carmen. 2010. http://www.cyta.com.ar. *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* [En línea] junio de 2010. http://www.cyta.com.ar/ta0502/v5n2a1.htm.

Méndez, Gonzalo. 2009. Ingeniería de Requisitos. [En línea] 2009. https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/03-Requisitos.pdf.

Moreno, Eduardo. 2017. Academia. [En línea] 2017. http://www.academia.edu/16516152/Microsoft_SQL_Server.

OACI. 2006. Anexo 1. *Licencias al Personal.* [En línea] 11 de marzo de 2006. http://www.udi.edu.co/images/biblioteca/aeronautica/anexo1.pdf.

Pena Aponte, Karla Vanesa. 2014. Slide Share. [En línea] 2014. https://de.slideshare.net/karlaaponte/sistema-gestor-de-base-de-datos-13334059.

pmoinformatica. 2015. pmoinformatica. [En línea] 2015. http://www.pmoinformatica.com/2015/05/requerimientos-no-funcionales-ejemplos.html.

Porto, Julián Pérez y Merino, María. 2014. Definicion.de. [En línea] 2014. http://definicion.de/licencia/.

PR Newswire. 2010. PR Newswire. [En línea] octubre de 2010. http://www.prnewswire.co.uk/news-releases/con-pycharm-los-desarrolladores-de-python-obtienen-finalmente-una-ide-potente-155001805.html.

—. 2011. PR Newswire. [En línea] 2011. http://www.prnewswire.com/news-releases/phpstorm-50-la-programacion-en-php-alcanza-su-maxima-expresion-169588156.html.

Pressman, Roger S. 2010. *Ingeniería de Software - Un enfoque práctico.* 2010.

Sánchez, Tamara Rodríguez. 2015. *Metodología de desarrollo para la actividad productiva de la UCI.* La Habana : s.n., 2015.

Sommerville, Ian. 2012. *Ingeniería del Software.* Madrid : PEARSON ADDISON WESLEY, 2012.

Sribd. 2016. Ventajas y desventajas de SGBD. [En línea] 2016. https://es.scribd.com/doc/89920963/Ventajas-y-Desventajas-de-SGBD.

Suarez, Montoya y María, Lina. 2013. *Enseñanza en la Ingeniería de Software: Aproximación a un Estado del Arte. Lámpsakos.* 2013.

Symfony. 2017. Librosweb. *Symfony 1.4, la guía definitiva.* [En línea] 2017. http://librosweb.es/libro/symfony_1_4/capitulo_15/pruebas_unitarias.html.

—. 2013. Symfony en Español. [En línea] 2013. http://gitnacho.github.io/symfony-docs-es/contributing/code/standards.html.

symfony. 2017. symfony.es. [En línea] 2017. http://symfony.es/.

Targetware Informática S.A.C. 2017. Software.com.ar. [En línea] enero de 2017. Targetware Informática S.A.C..

Téllez, Linda Luna. 2012. Scribd. [En línea] 2012. https://es.scribd.com/doc/91676941/Metodologias-agiles-vs-tradicionales.

TiendaLinux. 2016. TiendaLinux. [En línea] diciembre de 2016. https://soporte.tiendalinux.com/portal/Portfolio/postgresql_ventajas_html.

vyvquality. 2016. vyvquality.com. [En línea] 2016. http://vyvquality.com/pruebas-seguridad/.

Monsalve, E.; Werneck, V.; Leite, J. C. S. P. Evolución de un Juego Educacional de Ingeniería de Software a través de Técnicas de Elicitación de Requisitos. En *Proceedings of XIII Workshop on Requirements Engineering (WER'2010), Cuenca, Ecuador*. 2010. p. 12-23.

Quiñones, Martha Elena Vargas; DE VEGA, Luzángela Aldana. *Calidad y servicio: conceptos y herramientas*. Ecoe Ediciones, 2015.

Agudelo, Andrea López, et al. *DISEÑO DE UN PROTOTIPO PARA LA PROGRAMACIÓN DETALLADA Y DESPACHO DE LA PRODUCCIÓN BASADO EN EL ESTÁNDAR ISA-95*. *REVISTA GTI*, 2015, vol. 13, no 37, p. 81-85.

Toro, A.; Gálvez, J. G. Especificación de requisitos de software: una mirada desde la revisión teórica de antecedentes. *Entre Ciencia e Ingeniería*, 2016, no 19, p. 108-113.