

Facultad de Ingeniería Informática

Tema: Modelos para el estudio de las variables que más influyen en la estimación de la legitimidad en el modelo probabilístico de autenticación propuesto por Legón.

Informe de Gestión de proyecto

**Autor:** Estrella Cárdenas Blanco ([estrellacarden](hserra@ceis.cujae.edu.cu)asblanco@gmail.com)

**Tutor:** Dr. C. Raisa Socorro Llanes (<raisa@ceis.cujae.edu.cu>)

Rol Analista-Arquitecto-Diseñador

La Habana 2022

**Resumen**

En la modernidad los usuarios necesitan acceder a muchos servicios digitales imprescindibles para su vida cotidiana por lo que la autenticación de un usuario para concederle acceso a un sistema es un aspecto esencial para la seguridad de la información. Las contraseñas alfanuméricas son las más usadas pero tienen una contradicción entre su seguridad y su usabilidad, pues para ser seguras deben ser aleatorias, largas y no predecibles, mientras que para ser usables deben ser memorizables. Una de las alternativas de solución a este problema es la autenticación gráfica. Los sistemas de autenticación gráfica basan su funcionamiento en que los usuarios reconozcan patrones de imágenes o partes de ellas, en lugar de recordar largas y complejas secuencias de caracteres. Entre las técnicas de autenticación gráfica, *el PassPoint* ha recibido atención especial, es una de las técnicas basadas en la memorización de puntos como contraseña. Los modelos probabilísticos en autenticación gráfica se aplican para estimar la clave más probable a seleccionar, en cada imagen, por el usuario que se va a registrar. En este trabajo se desarrollará un sistema que registre los intentos de autenticación de los usuarios incorporando al esquema *PassPoint* en el modelo probabilístico implementado en la tesis del Dr. Legón.

Palabras claves: Autenticación Gráfica, Modelo Probabilístico, *PassPoint*.

**Abstract**

In modern times, users need access to many digital services essential for their daily lives, so authenticating a user to grant access to a system is an essential aspect for information security. Alphanumeric passwords are the most used but they have a contradiction between their security and their usability, because to be secure they must be random, long and not predictable, while to be usable they must be memorizable. One of the alternative solutions to this problem is graphical authentication. Graphical authentication systems base their operation on the fact that users recognize patterns of images or parts of them, instead of remembering long and complex sequences of characters. Among the graphical authentication techniques, the PassPoint has received special attention, it is one of the techniques based on the memorization of points as a password. The probabilistic models in graphical authentication are applied to estimate the most probable key to be selected, in each image, by the user to be registered. In this work, a system will be developed that records user authentication attempts, incorporating the PassPoint scheme into the probabilistic model implemented in Dr. Legón's thesis.

Keywords: Graphical Authentication, Probabilistic Model, PassPoint.

Índice General

[Contenido 1](#_Toc104887554)

[1. Documentación de Ingeniería de Requisitos: 1](#_Toc104887555)

[1.1 Breve descripción del negocio. 1](#_Toc104887556)

[1.2 Requisitos funcionales. 1](#_Toc104887557)

[1.3 Requisitos no funcionales. 1](#_Toc104887558)

[1.4 Diagrama de casos de uso del sistema. 2](#_Toc104887559)

[1.5 Descripción de alto nivel de todos los casos de usos representados. 3](#_Toc104887560)

[2. Definir Arquitectura Candidata 4](#_Toc104887561)

[2.1 Estudio y selección de la tecnología 4](#_Toc104887562)

[2.2 Tecnologías a utilizar en la solución 12](#_Toc104887563)

[3. Identificar y justificar los estilos y patrones de arquitectura que se proponen para su proyecto. 13](#_Toc104887564)

[3.1 Representar cada uno de los patrones Arquitectónicos que se identificaron. 13](#_Toc104887565)

[4. Identificar patrones y principios de diseño a emplear en la solución. De cada patrón: 15](#_Toc104887566)

[4.1 Uso del patrón (¿dónde se usa?) 15](#_Toc104887567)

[4.2 Justificación o motivo del patrón. (¿Por qué se usa?) 16](#_Toc104887568)

[4.3 Diagrama de clases donde muestren cómo se implementa el patrón en su propuesta 16](#_Toc104887569)

[4.4 Principios de diseño empleados. Mencione en que parte del de la propuesta de solución se tiene en cuenta. 17](#_Toc104887570)

[5. Estándar de codificación 17](#_Toc104887571)

[6. Modelo de despliegue 18](#_Toc104887572)

[7. Pruebas funcionales 19](#_Toc104887573)

[7.1 Casos de prueba 19](#_Toc104887574)

[Referencias 29](#_Toc104887575)

Índice de Figuras

[Figura 1 Diagrama de Caso de Uso del Sistema 6](#_Toc104320227)

[Figura 2 Modelo Vista Controlador 17](#_Toc104320228)

[Figura 3 Arquitectura de capas basado en responsabilidades 17](#_Toc104320229)

[Figura 4 Filtro y Tubería 18](#_Toc104320230)

[Figura 5 Patrón orientado a servicios 18](#_Toc104320231)

[Figura 6 Patrón Experto 19](#_Toc104320232)

[Figura 7 Patrón Singleton 19](#_Toc104320233)

[Figura 8 Diagrama del despliegue 21](#_Toc104320234)

Índice de las Tablas

[Tabla 1 Requisitos no funcionales 5](#_Toc104320213)

[Tabla 2 Caso de Uso: Mostrar reportes del usuario estándar 6](#_Toc104320214)

[Tabla 3 Caso de Uso: Autenticar Usuario 6](#_Toc104320215)

[Tabla 4 Caso de Uso: Gestionar Imágenes 6](#_Toc104320216)

[Tabla 5 Caso de Uso: Gestionar Usuarios 7](#_Toc104320217)

[Tabla 6 Caso de Uso: Mostrar reportes del usuario administrador 7](#_Toc104320218)

[Tabla 7 Caso de Uso: Registrar contraseña 7](#_Toc104320219)

[Tabla 8 Caso de Prueba checkUser 22](#_Toc104320220)

[Tabla 9 Caso de Prueba report1 24](#_Toc104320221)

[Tabla 10 Caso de prueba report2 25](#_Toc104320222)

[Tabla 11 Caso de prueba report3 26](#_Toc104320223)

[Tabla 12 Caso de prueba report4 27](#_Toc104320224)

[Tabla 13 Caso de prueba report5 28](#_Toc104320225)

[Tabla 14 Caso de prueba de caja blanca 29](#_Toc104320226)

# Contenido

## 1. Documentación de Ingeniería de Requisitos:

### 1.1 Breve descripción del negocio.

Es una aplicación de autenticación de usuarios, donde para acceder al sistema los usuarios tienen que estar creados en la base de datos por el administrador, una vez estén registrados en la base de datos, cuando van a entrar al sistema sino tienen creada ninguna contraseña, crea la contraseña escogiendo una imagen de las que se le muestra seleccionando 5 puntos y luego crea la pregunta de seguridad, este proceso se denomina fase de registro y se puede repetir hasta 5 veces. Si un usuario tiene creada sus contraseñas se muestra una de las imágenes utilizadas aleatoriamente para que seleccione los 5 puntos de la imagen que selecciono en la fase de registro con un margen de error, este proceso se denomina fase de autenticación. Si todos los puntos se encuentran dentro de la región de tolerancia correspondiente entonces el usuario se autentica satisfactoriamente.

### 1.2 Requisitos funcionales.

A continuación se muestran los requisitos funcionales: La interesada que lo solicito es Raisa Socorro.

El sistema debe gestionar:

1- Autenticar usuario

2- Registrar contraseña de un usuario

3- Gestionar a los usuarios en la base de datos

4- Gestionar a las imágenes en la base de datos

5- Mostrar reportes

### 1.3 Requisitos no funcionales.

A continuación en la tabla 1 se muestran los requisitos no funcionales

|  |  |  |
| --- | --- | --- |
| Requisito | Problema frecuente | Descripción |
| Apariencia o Interfaz externa | Control de navegación del usuario  Control de eventos generados por el usuario | El sistema debe saber en todo momento donde se encuentra el usuario y hacia donde se puede dirigir dentro de la aplicación, así como ser capaz de manejar las excepciones que se lancen de forma tal que no ocurra perdida de datos o un bloqueo del sistema. |
| Rendimiento | Implementación del acceso a los datos  Registro de eventos diarios del sistema | El sistema debe funcionar de manera eficiente ya que tiene que garantizar que el tiempo de respuesta en la autenticación sea menor que un valor dado. Valor que se define normalmente en el tema de seguridad en el orden de los milisegundos. |
| Confiabilidad | Tolerancia a fallas | El sistema debe permitir la recuperación en caso de error y, de ser posible, mostrar la causa del mismo. |

Tabla 1 Requisitos no funcionales

### 1.4 Diagrama de casos de uso del sistema.

A continuación en la figura 1 se muestra el diagrama de caso de uso del sistema.

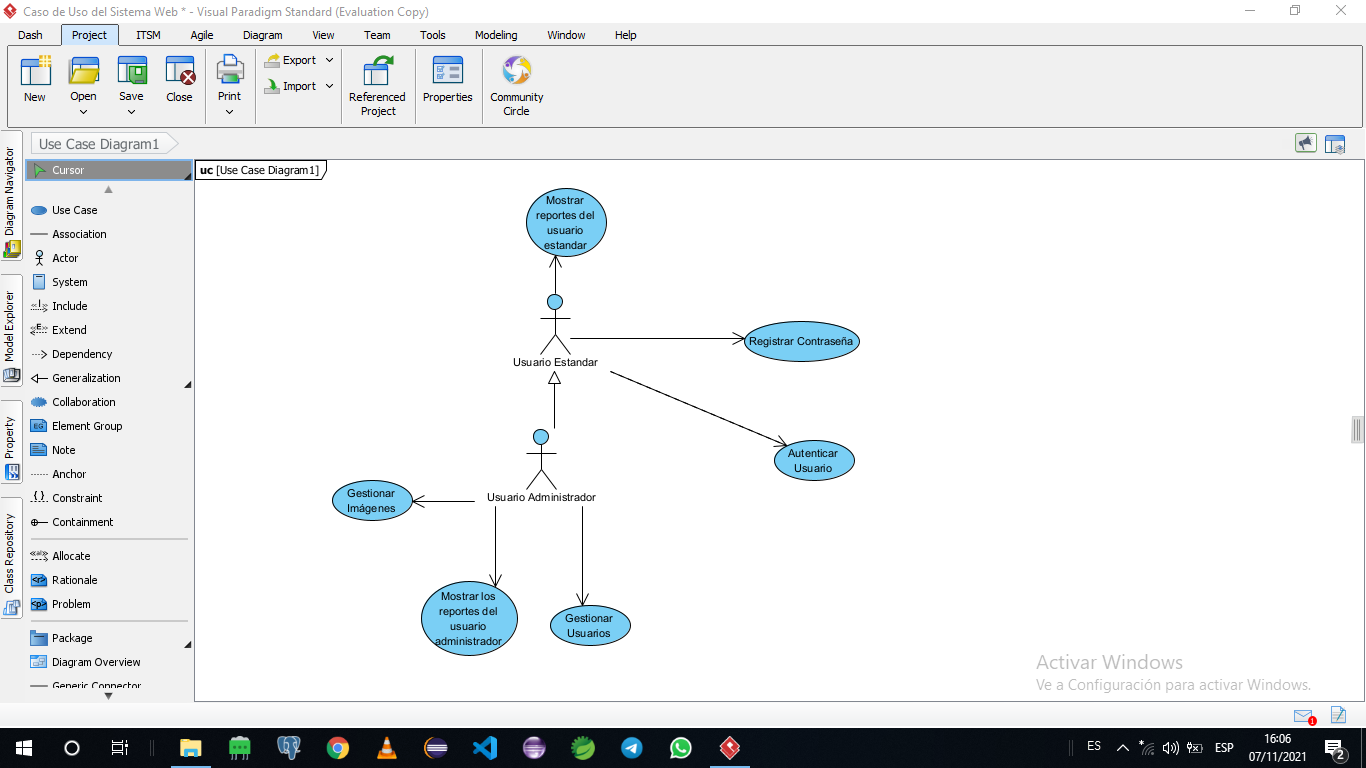


Figura 1 Diagrama de Caso de Uso del Sistema

### 1.5 Descripción de alto nivel de todos los casos de usos representados.

Tabla 2 Caso de Uso: Mostrar reportes del usuario estándar

|  |  |
| --- | --- |
| **Caso de uso** | Mostrar reportes del usuario estándar |
| **Actores** | Usuario Estándar |
| **Descripción**  El caso de uso comienza después que se haya autenticado el usuario y luego se muestran los reportes. | |
| **Requisitos funcionales** | 5- Mostrar reportes |

Tabla 3 Caso de Uso: Autenticar Usuario

|  |  |
| --- | --- |
| **Caso de uso** | Autenticar Usuario |
| **Actores** | Usuario Estándar y Administrador |
| **Descripción**  El caso de uso comienza una vez el usuario este registrado en la base de datos y ya haya creado su contraseña. | |
| **Requisitos funcionales** | 1- Autenticar usuario |

Tabla 4 Caso de Uso: Gestionar Imágenes

|  |  |
| --- | --- |
| **Caso de uso** | Gestionar Imágenes |
| **Actores** | Usuario Administrador |
| **Descripción**  El caso de uso comienza una vez el usuario se haya autenticado y puede que no gestione las imágenes. | |
| **Requisitos funcionales** | 4- Gestionar a las imágenes en la base de datos |

Tabla 5 Caso de Uso: Gestionar Usuarios

|  |  |
| --- | --- |
| **Caso de uso** | Gestionar Usuarios |
| **Actores** | Usuario Administrador |
| **Descripción**  El caso de uso comienza una vez el usuario se haya autenticado y puede que no se gestione los usuarios. | |
| **Requisitos funcionales** | 3- Gestionar a los usuarios en la base de datos |

Tabla 6 Caso de Uso: Mostrar reportes del usuario administrador

|  |  |
| --- | --- |
| **Caso de uso** | Mostrar reportes del usuario administrador |
| **Actores** | Usuario Administrador |
| **Descripción**  El caso de uso comienza después que se haya autenticado el usuario y luego se muestran los reportes. | |
| **Requisitos funcionales** | 5- Mostrar reportes |

Tabla 7 Caso de Uso: Registrar contraseña

|  |  |
| --- | --- |
| **Caso de uso** | Registrar contraseña |
| **Actores** | Usuario Administrador y Estándar |
| **Descripción**  El caso de uso comienza cuando el usuario entra por primera vez al sistema y se crea su contraseña. | |
| **Requisitos funcionales** | 2- Registrar contraseña de un usuario |

## 2. Definir Arquitectura Candidata

### 2.1 Estudio y selección de la tecnología

El objetivo de este trabajo es implementar un sistema de autenticación gráfica con el modelo propuesto por la tesis de Legón [[1](#_ENREF_1)] para validar que el modelo sea eficiente, ya que no se cuenta con ninguna implementación del modelo que pudiera registrar los intentos de autenticación de los usuarios que pudiera estimar la distribución de probabilidad del estadígrafo a partir de estos datos.

Si necesitas implementar una aplicación web y aún no sabes cual framework backend escoger, a continuación te presentamos los mejores framework backend.

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Característica | Lenguaje | Curva de Aprendizaje |
| Laravel | * Código simple y elegante. * Sistema de rutas muy completo. * Leve curva de aprendizaje. * Utiliza Blade como sistema de plantillas. * Utiliza Eloquent para el manejo de la abstracción a la capa de datos. * Permite el uso de middlewares que pueden ejecutarse antes o después que una acción específica. * Posee un servidor de desarrollo. * Cuenta con una amplia comunidad y una documentación muy completa, lo que se traduce en soporte para cualquier problema en el proceso de desarrollo. * Estructura de programación intuitiva. * Tiene implementados los procesos de autenticación y autorización. | PHP | Leve |
| Symfony | * Sigue buenas prácticas de codificación. * Uso intenso de patrones en su arquitectura. * Arquitectura Modelo – Vista – Controlador. * Curva de aprendizaje media. * Utiliza Twig como motor de plantillas. * Posee un servidor de desarrollo. * Utiliza Doctrine para el manejo de la abstracción a la capa de datos. * Cuenta con una amplia comunidad y una documentación muy completa. * Permite y fomenta la realización de pruebas unitarias. * Código abierto con licencia MIT. * Protección contra ataques CSRF o XSS. | PHP | Media |
| Django | * Sistema de código abierto con licencia BSD. * Cuenta con un sistema de administración listo para ser utilizado sin necesidad de configuración. * Protección contra ataques CSRF e inyecciones SQL. * Posibilita la creación de sistemas escalables utilizando su modularidad. * Comunidad extensa y amplia documentación. * Posee un servidor de desarrollo. * Con su librería Django Rest Framework, el proceso de creación de API Rest es muy sencillo y rápido. * Procesos de autenticación y autorización implementados de modo sencillo. * Control de los permisos a tal punto de decir que usuario puede o no crear, editar, ver y eliminar registros de un modelo específico. * Utilizado en webs como: Instagram, Spotify, Pinterest, The Washington Post, entre otras. | Python | Leve |
| Flask | * Arquitectura Modelo – Vista – Controlador. * Licencia BSD. * Posee un servidor de desarrollo. * Contiene un depurador y soporte para pruebas unitarias. * Soporta de manera nativa el uso de cookies seguras. * Rendimiento óptimo. * No cuenta con ORM integrado por defecto. * Amplia documentación. * Diseño minimalista. * Utilizado en webs como: Netflix, RedHat, Reddit entre otras. | Python | Leve |
| Spring | * Se basa en la filosofía Convención sobre configuración (CoC). * No impone arquitecturas, pero es perfectamente compatible con Modelo – Vista – Controlador. * Es completamente modular. * Gestión de transacciones. * Herramientas para pruebas integradas. * Permite la implementación de rutinas transversales. * Enfocado en la seguridad de tu sistema. * Comunidad activa. * Curva de aprendizaje elevada en comparación con algunos de los frameworks de este listado. | Java | Alta |
| Express | * Código abierto. * Licencia MIT. * Estructura básica y muy flexible. * Excelente rendimiento y tiempos de respuesta. * Con Express puedes implementar webs utilizando solamente Javascript. * Permite crear API fácilmente. * Manejo de rutas. * Permite el uso de middlewares. * Utilizado en Fox Sports, PayPal, Uber, IBM, entre otros. | JavaScript | Leve |
| Asp.net Core | * Arquitectura Modelo – Vista – Controlador. * De código abierto y centrado en la comunidad. * Integración con los principales [framework frontend](https://saasradar.net/blog/mejores-framework-frontend/) modernos. * Capacidad para desarrollarse y ejecutarse en Windows, macOS y Linux. * Programación mínima, gran cantidad de funcionalidad básica implementada. * Alto rendimiento. * Capacidad de hospedarse en IIS, Nginx, Apache o Docker. * Control de versiones en paralelo. * Ofrece a los desarrolladores la opción de usar patrones de programación asíncrona. | C# | Leve |

A continuación se muestra las características y beneficios de implementar Spring Framework[[2](#_ENREF_2)]:

Características:

Resulta imposible hablar de un Framework sin enumerar sus características. En el caso de Spring podemos encontrar las siguientes que ofrecen una cantidad considerable de servicios:

* **Tecnologías:** Spring es completamente modular y soporta diferentes tecnologías como la inyección de dependencias, eventos, recursos, i18n, validación, enlace de datos, conversión de tipo, SpEL.
* **Acceso a datos:** soporte DAO, JDBC, ORM, Marshalling XML.
* **Gestión de transacciones.**
* **Integración:** comunicación remota, JMS, JCA, JMX, correo electrónico, tareas, programación, caché.
* **Pruebas (Testing):** simulacro de objetos, el framework TestContext, Spring MVC prueba, WebTestClient.
* **Programación orientada a aspectos (AOP):** permite la implementación de rutinas transversales.
* Facilita en gran medida la programación basada en **MVC (Modelo Vista Controlador)** y una implementación rápida basada en **Inyección de Dependencias (Dependency Injection).**
* Es un Framework que tiene un especial foco sobre la **Seguridad.**
* Compatible con **Frameworks web:** Spring WebFlux y Spring MVC.
* **Permite el procesamiento de datos por lotes.**
* **Administración Remota:** a través de este módulo se puede configurar la visibilidad y gestión de los objetos Java para la configuración local o remota vía JMX.
* Es un **framework liviano** debido a su implementación POJO (Plain Old Java Object), Spring Framework no obliga al programador a heredar ninguna clase ni a implementar ninguna interfaz.

Beneficios de utilizar Spring Framework

Si bien las características de Spring son comunes a la mayoría de los Framework actuales siempre surge la pregunta sobre **cuáles son los beneficios de utilizar Spring** frente otras alternativas. Quizá la más significativa se encuentra en su propio diseño modular. Esto hace que a pesar de la cantidad de paquetes y clases que tiene podemos ocuparnos de aquello que necesitamos ignorando el resto.

Esto hace también que **testear nuestra aplicación escrita con Spring** sea un proceso simple. Nuestro código dependiente se traslada a este framework de modo transparente al usuario y mediante el uso de “JavaBeanStyle” la inyección de dependencias para pruebas se puede realizar de una manera sencilla mediante la introducción de datos dummies o mocks.

Por supuesto, como decía, **Spring es un framework basado en MVC** para web muy bien diseñado que proporciona estabilidad a nuestro código pero quizá su punto fuerte es que proporciona una API para traducir excepciones específicas de la tecnología (como por ejemplo las generadas por JDBC, Hibernate o JDO) en excepciones consistentes y no verificadas. Pero además Spring proporciona una interfaz de gestión de transacciones coherente que puede reducirse a una transacción local (utilizando una única base de datos) y ampliarse a transacciones globales.

En lo personal también destacaría la **posibilidad de usar plantillas** para diversas tecnologías entre la cuales podemos destacar las siguientes: JDBC, Hibernate y JPA, de forma tal que **no hay necesidad de escribir un código extenso**, ya que con estas plantillas simplifica el trabajo en cuanto a los pasos básicos a implementar de estas tecnologías. Por lo que desarrollar una aplicación resulta en una tarea bastante llevadera en comparación a si lo hacemos con otras alternativas.

Por si esto fuera poco, ya por último, **Spring cuenta con una gran comunidad muy activa** que genera una ingente cantidad de documentación lo que facilita y mucho nuestra curva de aprendizaje o resolución de dudas. Especialmente si te estás iniciando en este framework o si estás realizando una migración de tu proyecto. Todo esto lo convierte en una opción a tener en cuenta, si no lo has hecho ya, para un desarrollo de software saludable y eficiente.

**¿Por qué usar Spring?**

Aunque no sea una característica única de Spring, el uso de inyección de dependencias facilita la programación contra interfaz, permitiendo a los distintos componentes depender únicamente de interfaces y produciendo así un **código menos acoplado** y acorde al Principio Abierto/Cerrado (Open/Closed Principle). Además, permite implementar el patrón *singleton* de una forma extremadamente sencilla (por defecto, las dependencias que se inyectan son singletons).

Hay cientos de tecnologías que Spring permite integrar. Desde bibliotecas que implementan [opentracing](https://opentracing.io/) hasta las que generan métricas para una aplicación, pasando por serialización/deserialización a JSON y XML, seguridad con OAuth2 o programación reactiva, entre otras.

En general, Spring **aumenta la productividad y reduce la fricción** al ofrecer abstracciones sobre implementaciones de tecnologías concretas. Un ejemplo claro es el de spring-data, que permite definir el acceso a base de datos con interfaces Java. Esto lo consigue *parseando* el nombre de los métodos y generando la consulta con la sintaxis específica para el *driver* que se utilice. Por ejemplo, cambiar una aplicación de MySQL a PostgreSQL es tan sencillo como cambiar el *driver*: Spring se encarga de la sintaxis de forma transparente.

A pesar de "la magia de Spring", como muchos lo llaman, Spring permite desactivar estos "comportamientos mágicos" en caso de ser necesario, por lo que **es posible tomar el control cuando se necesite más granularidad**. Siguiendo con el ejemplo de spring-data, este control sería necesario si se tienen que realizar consultas mucho más complejas que un SELECT \* BY name. En esos casos, entre otras opciones, es posible anotar el método con @Query y escribir la consulta que se dese. De esta manera “desaparece la magia”.

Por lo general, Spring no obliga a implementar ni extender nada, lo que permite es escribir código que es "agnóstico" del *framework*. De esta forma, desarrolladores con cero o muy poco conocimiento de Spring pueden realizar su trabajo sin mayores complicaciones.

### 2.2 Tecnologías a utilizar en la solución

Para el desarrollo de cualquier sistema de software es necesaria la utilización de tecnologías que permitan facilitar el trabajo, para resolver la problemática se pretende construir una aplicación web, a continuación se exponen las tecnologías seleccionadas.

**Spring Tool 4:** Diseñado para desarrollar aplicaciones empresariales usando Spring Framework y Spring Boot, la nueva generación de Spring Tools brinda soporte de desarrollo de clase mundial para sus aplicaciones Spring. Nuestras herramientas tienen un profundo conocimiento de Spring integrado.

**Spring Boot:** Proporciona una buena plataforma para que los desarrolladores de Java desarrollen una aplicación de resorte de grado de producción que puede ejecutar. Puedes empezar con configuraciones mínimas sin la necesidad de un conjunto completo de configuración de Spring [[3](#_ENREF_3)].

**Spring Framework:** Spring Framework es una solución liviana y una ventanilla única potencial para construir su aplicación lista para la empresa. Sin embargo, Spring es modular, lo que le permite usar solo aquellas partes que desee, sin necesidad de tener que traer el resto. Puede utilizar el contenedor de IoC, con cualquier marco web, pero también puede usar solo el código de integración de Hibernate o la capa de abstracción JDBC. La primavera Framework admite la gestión declarativa de transacciones, el acceso remoto a su lógica a través de RMI o servicios web y varias opciones para conservar sus datos. Ofrece un marco MVC con todas las funciones y le permite integrar AOP de forma transparente en su software [[2](#_ENREF_2)].

**Primefaces:** Es una suite de componentes JSF de código abierto con varias extensiones [[4](#_ENREF_4)].

• Amplio conjunto de componentes (HTML Editor, Dialog, Autocomplete, Charts y muchos más).

• Ajax integrado basado en API JSF Ajax estándar.

• Ligero, sin configuración y sin dependencias requeridas.

• Skinning Framework con más de 35 temas incorporados y soporte para la herramienta de diseño de temas visuales.

• Temas y diseños premium

• Amplia documentación.

• Comunidad de usuarios grande, vibrante y activa.

• Desarrollado con "pasión" desde desarrolladores de aplicaciones hasta desarrolladores de aplicaciones.

**PostgreSQL:** PostgreSQL es un potente sistema de base de datos relacional de objetos de código abierto con muchos años de desarrollo activo que le ha ganado una sólida reputación por su fiabilidad, solidez de funciones y rendimiento [[5](#_ENREF_5)].

## 3. Identificar y justificar los estilos y patrones de arquitectura que se proponen para su proyecto.

Estilo: llamada y retorno, Patrón: Modelo Vista Controlador

Estilo: Llamada y retorno, Patrón: Arquitectura de capas basado en responsabilidades

Estilo: Flujo de datos, Patrón: Filtros y Tubería

Estilo: Peer to Peer, Patrón: Arquitectura Orientado a servicios

### 3.1 Representar cada uno de los patrones Arquitectónicos que se identificaron.

A continuación en la figura 2 se muestra el modelo vista controlador

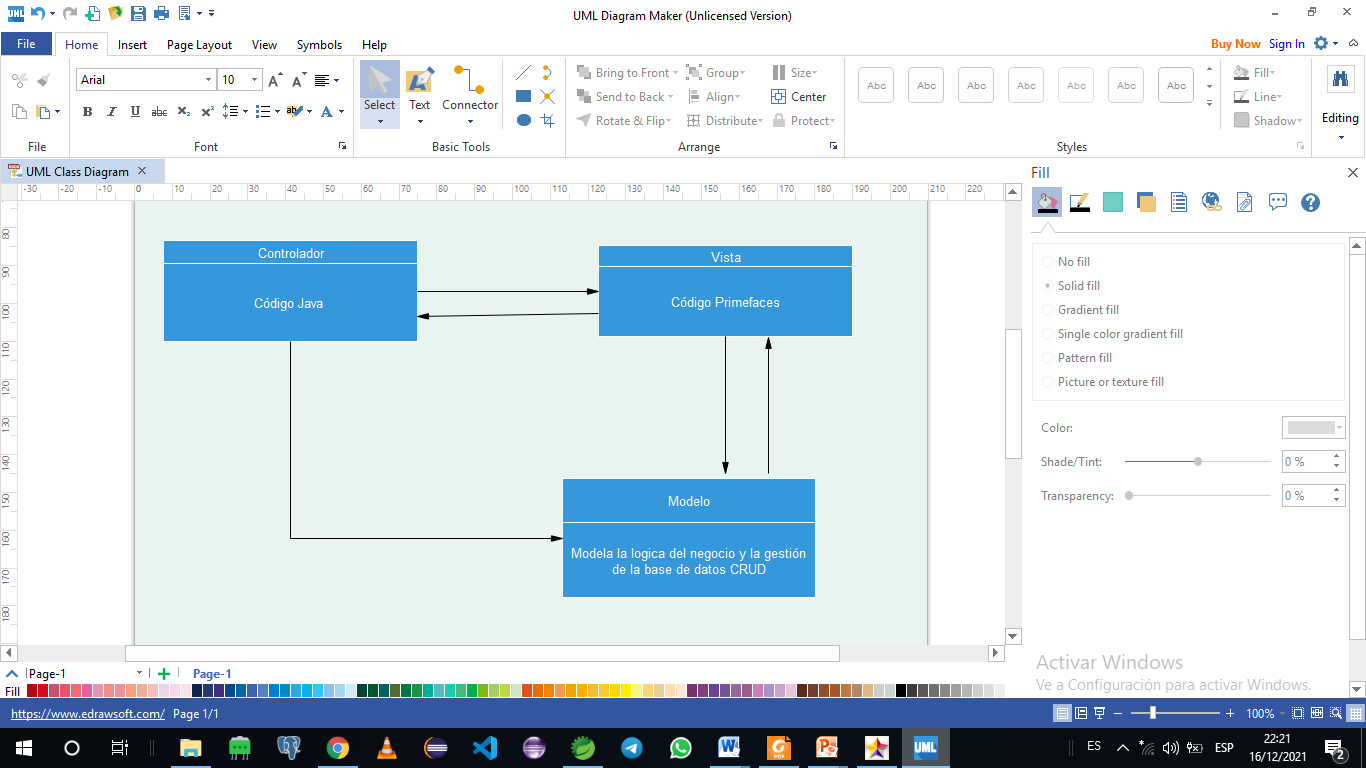


Figura 2 Modelo Vista Controlador

A continuación en la figura 3 se muestra la arquitectura de capas basado en responsabilidades

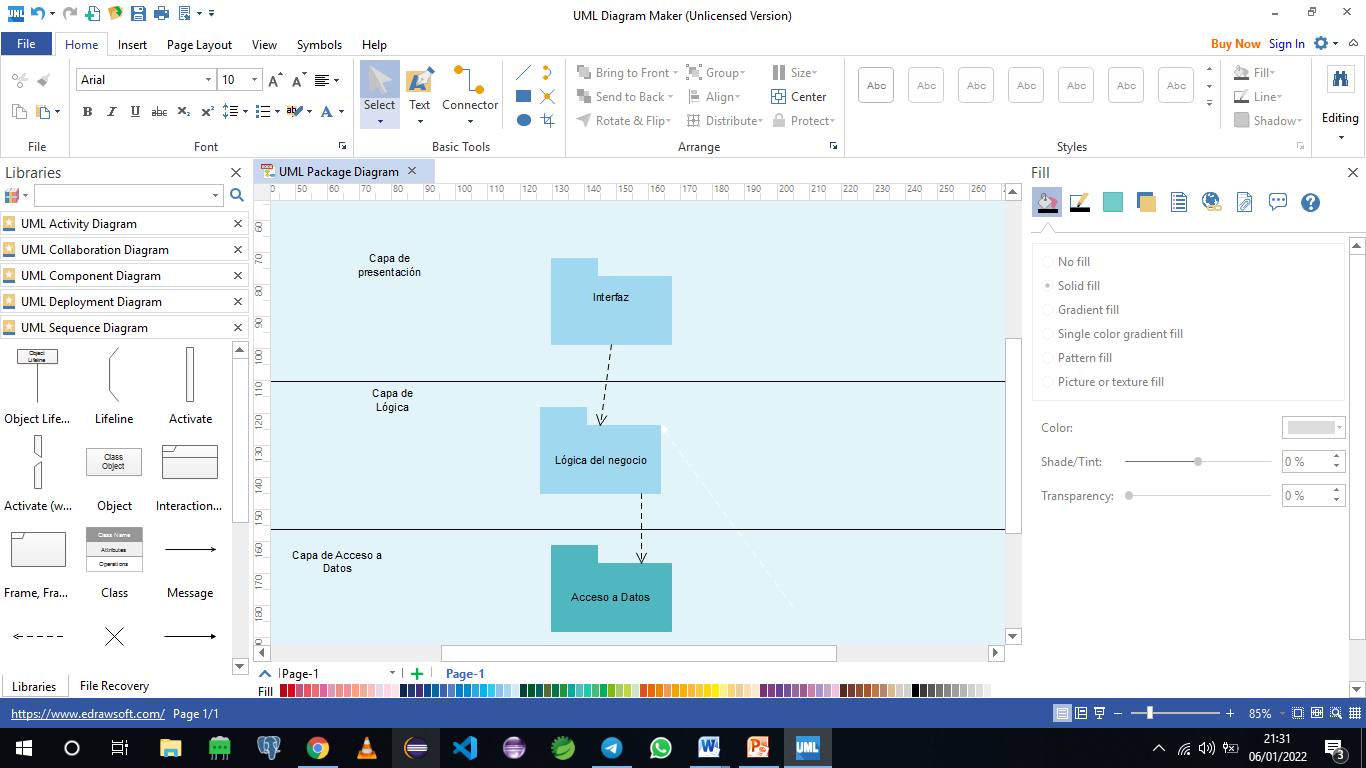


Figura 3 Arquitectura de capas basado en responsabilidades

A continuación en la figura 4 se muestra el patrón Filtros y Tuberías del proceso de determinar si un usuario es legítimo, ilegítimo o dudoso legítimo.

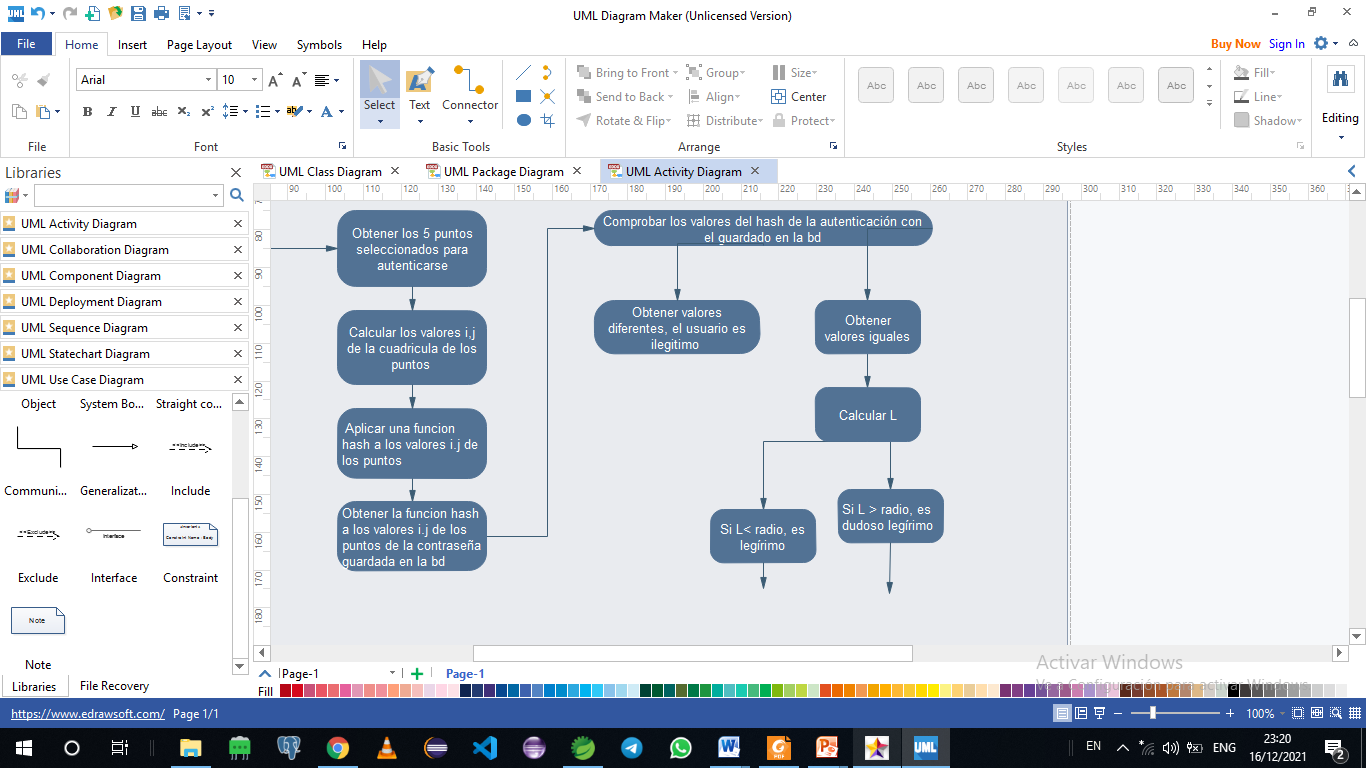


Figura 4 Filtro y Tubería

A continuación en la figura 5 el patrón Arquitectura Orientado a servicios. El Apache Tomcat es un ejemplo de servidor stateful. Un servidor stateful puede recordar la información entre las peticiones.

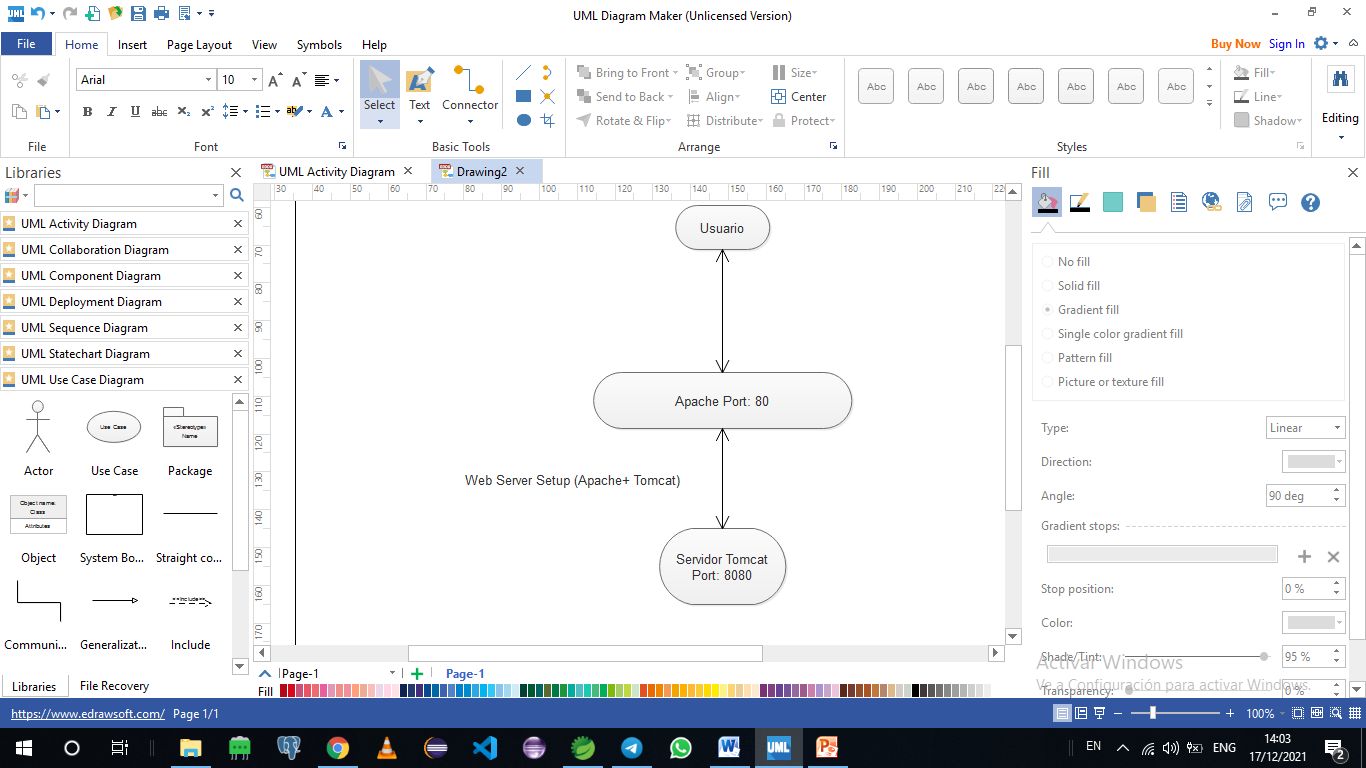


Figura 5 Patrón orientado a servicios

## 4. Identificar patrones y principios de diseño a emplear en la solución. De cada patrón:

### 4.1 Uso del patrón (¿dónde se usa?)

Patrón Experto: Se usa en todas las clases, cada clase cuenta con la información necesaria para cumplir con su responsabilidad.

Patrón Singleton: Lo que hace Spring al detectar un bean singleton es crear una sola instancia por contexto de aplicación.

### 4.2 Justificación o motivo del patrón. (¿Por qué se usa?)

Patrón Experto: Los objetos hacen cosas relacionadas con la información que posee.

Patrón Singleton: Consiste en mantener una única instancia del objeto a lo largo de toda la aplicación.

### 4.3 Diagrama de clases donde muestren cómo se implementa el patrón en su propuesta

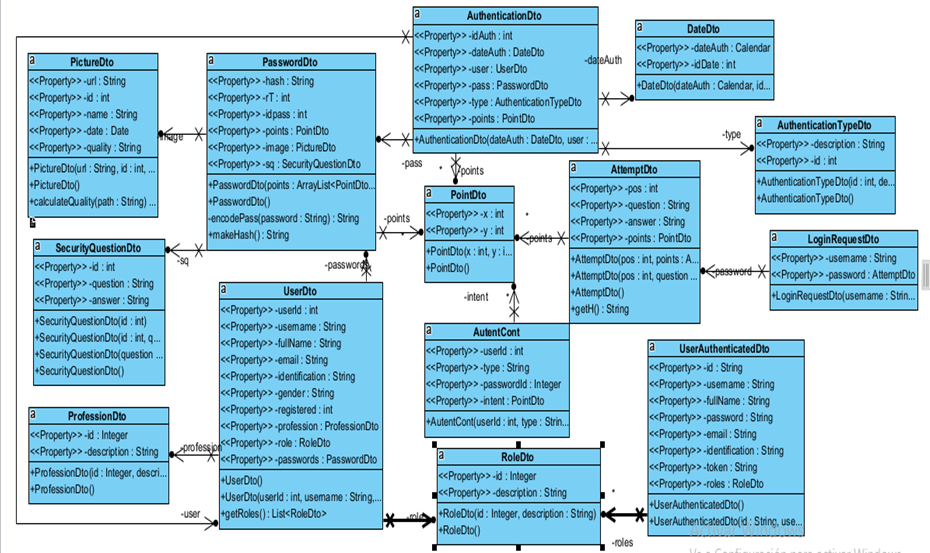


Figura 6 Patrón Experto

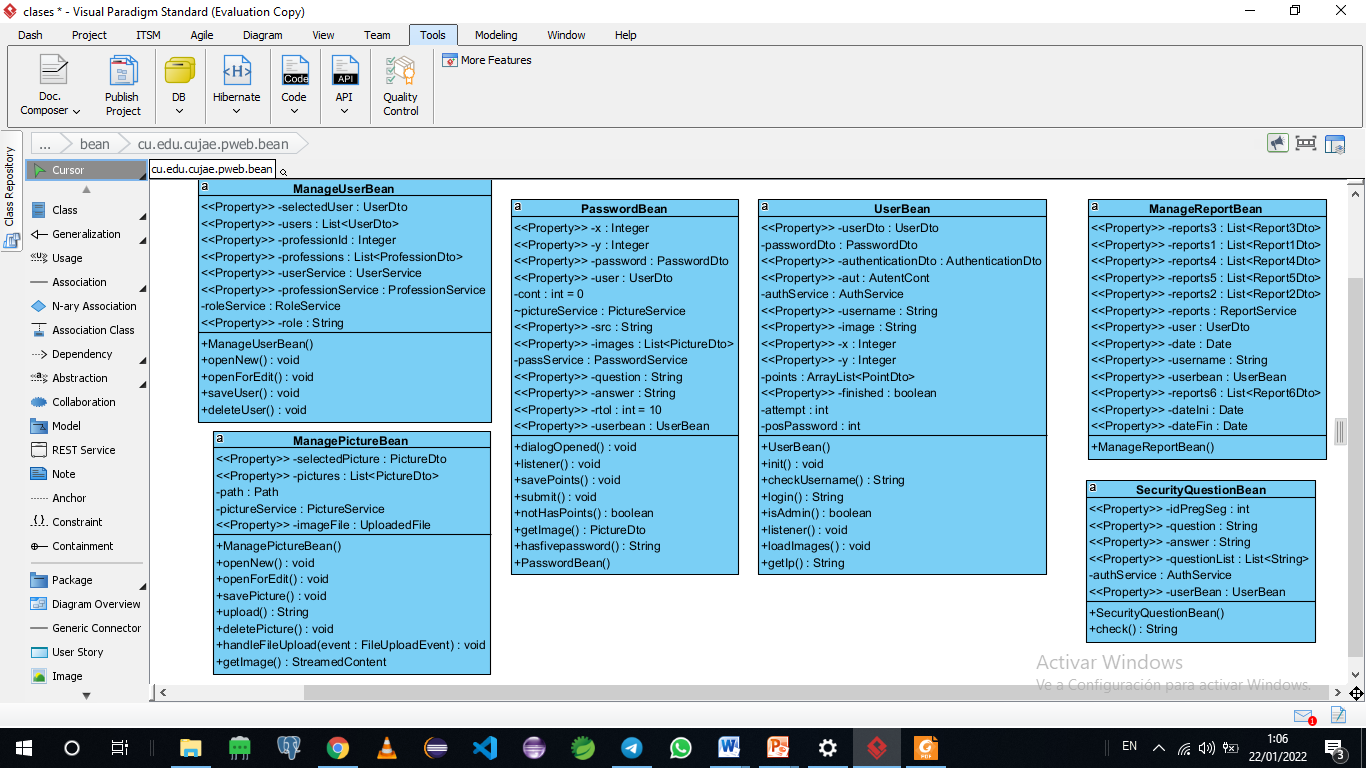


Figura 7 Patrón Singleton

### 4.4 Principios de diseño empleados. Mencione en que parte del de la propuesta de solución se tiene en cuenta.

Principio Única responsabilidad: Solo existe un objeto que se instancia una sola vez por lo que solo se maneja el método que crea la instancia de la clase (constructor) y otro que evita que se creen nuevos objetos.

Diseñar hacia las interfaces, no hacia las implementaciones: En el sistema los servicios de acceso a la base de datos están implementados con interfaces y luego se crean clases que implementen esas interfaces y crean los métodos correspondientes.

Ley de Demeter: La clase Usuario se relacionan únicamente con la clase Contraseña minimizando el impacto de los cambios.

Principio Hollywood: Se ha diseñado una pieza de alto nivel (Usuario) que interactúa con piezas de más bajo nivel (resto de las contenidas en el subsistema) hacienda el papel de coordinadora.

## 5. Estándar de codificación

Para que estas cosas no pasen y tu código siempre esté a la altura, debemos seguir siempre un estándar de codificación. Los estándares de código, son parte de las llamadas buenas practicas o mejores prácticas, estas son un conjunto no formal de reglas, que han ido surgiendo en las distintas comunidades de desarrolladores con el paso del tiempo y las cuales, bien aplicadas pueden incrementar la calidad de tu código, notablemente.

Entendemos como estándar de código a un conjunto de convenciones establecidas de ante mano (denominaciones, formatos, etc.) para la escritura de código. Estos estándares varían dependiendo del lenguaje de programación elegido y además varían en cobertura, algunos son más extensos que otros. Pero hay puntos que todos estándares deberían cubrir:

**Comentarios**

Aunque muchas veces, a todos nos da pereza escribirlos, porque pensamos que nuestro código se explica solo, los comentarios son una parte fundamental de un buen código fuente, esto es por varias razones, entre ellas está el explicar ese trocito de código que has condensado en 10 líneas, utilizando ternarios y diferentes funciones ayudará a un futuro tu o a un compañero que tenga que revisar tu código a entender más fácilmente lo que querías conseguir. Además de los comentarios para explicar determinadas partes de tu código, también es recomendable utilizar bloques de documentación en funciones, explicando parámetros, tipos esperados y lo que se persigue conseguir.

**Convenio de denominaciones o naming conventions.**

Otro de los puntos a tener en cuenta a la hora de aplicar unos buenos estándares de código, es el convenio de denominación. Estos convenios nos darán una guía clara de cómo debemos elegir los nombres de clases, funciones y variables. Muchos programadores suelen utilizar por comodidad nombres de variables que no tienen significado alguno ($var, int X, string palabra, etc.) y que solo hacen que entorpecer el trabajo de otros miembros del equipo o de ellos mismos cuando tienen que revisar su código después de un tiempo sin tocarlo.

**Legibilidad – Formato y espaciados**

Por último, un buen código ha de ser fácil de leer y tener una buena estructuración visual que acompañe a la lógica del algoritmo en cuestión. A propósito de cómo ha de hacerse, normalmente cada estándar define una serie de reglas sobre cómo han de posicionarse las llaves, número de espacios a utilizar para tabular, longitud de las líneas, etc.

## 6. Modelo de despliegue

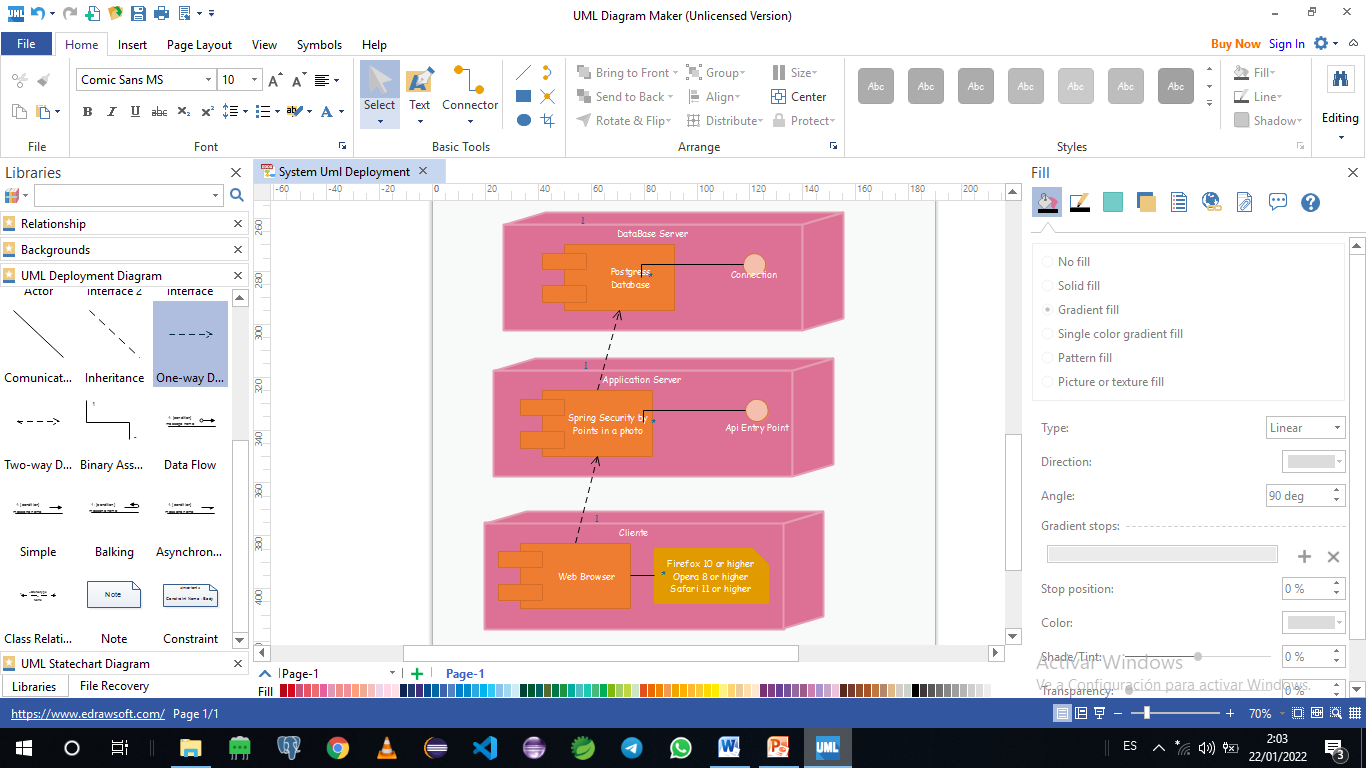


Figura 8 Diagrama del despliegue

## 7. Pruebas funcionales

Las pruebas de caja negra, también conocidas como pruebas de comportamiento o pruebas funcionales, tienen como objetivo chequear que los sistemas desarrollados cumplan con sus funcionalidades. Se enfocan solamente en la entrada y la salida del sistema sin tener en cuenta la estructura interna del software. Es recomendable ejecutar los escenarios con entradas válidas y fallidas [[6](#_ENREF_6)].

Las pruebas de caja blanca, también conocidas como pruebas de cristal o pruebas estructurales, es una técnica que evalúa el código y la estructura interna del programa [[6](#_ENREF_6)].

### 7.1 Casos de prueba

Con el objetivo de comprobar el correcto funcionamiento del sistema de software se proponen casos de pruebas guiados a los casos de uso.

Crear autenticación: Verificar que el usuario existe en la base de datos para entrar al sistema.

Exportar Reportes: Verificar que se muestren correctamente los reportes

Gestionar Imágenes: Verificar que se inserte, modifique y elimine las imágenes.

Gestionar Usuarios: Verificar que se inserte, modifique y elimine los usuarios.

A continuación se muestras las pruebas de caja negras realizadas

Tabla 8 Caso de Prueba checkUser

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Historia de Usuario** | | | checkUser | | | |
| **Desarrollador** | | | Estrella Cárdenas | | | |
| **Probador** | | | Estrella Cárdenas | | | |
| **Fecha** | | | **29/10/2019** | | | |
| **Versión** | | | **1.0** | | | |
| **Objetivo de la Prueba** | | | Comprobar que funcione correctamente | | | |
| **Descripción de la prueba** | | | Permite verificar cuando un usuario se va autenticar si existe o no en la base de datos. Si existe el usuario puede entrar al sistema. | | | |
| **Condiciones** | | | Existir en la base de datos | | | |
| **Combinaciones de valores de entrada** | | | | | **Resultados Esperados** | **Resultados Reales** |
| **CP** | **escenario** | **nombre variable de entrada** | | **valor** |
| 1 checkUser | | username | | star | Devuelve un objeto UserDto ya que el usuario star existe en la base de datos y sus atributos son:  Userid=1  Identification = 00060566498  Gender=femenino  Profid=1  Username= star  Fullname= estrella  Registered=1  Roleid=1  Email= estrellacardenas@gmail.com | Devuelve un objeto UserDto ya que el usuario star existe en la base de datos y sus atributos son:  Userid=1  Identification = 00060566498  Gender=femenino  Profid=1  Username= star  Fullname= estrella  Registered=1  Roleid=1  Email= estrellacardenas@gmail.com |
| 2 | checkUser | username | | juan | Devuelve un objeto UserDto = null ya que el usuario juan no existe en la base de datos | Devuelve un objeto UserDto = null ya que el usuario juan no existe en la base de datos |
|  | | | | | | |

Tabla 9 Caso de Prueba report1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Historia de Usuario** | | | Report1 | | | |
| **Desarrollador** | | | Estrella Cárdenas | | | |
| **Probador** | | | Estrella Cárdenas | | | |
| **Fecha** | | | **29/10/2019** | | | |
| **Versión** | | | **1.0** | | | |
| **Objetivo de la Prueba** | | | Comprobar que funcione correctamente | | | |
| **Descripción de la prueba** | | | Resumen de los intentos de autenticación de todos los usuarios según el tipo en un día dado. | | | |
| **Condiciones** | | | Existir autenticaciones en la base de datos | | | |
| **Combinaciones de valores de entrada** | | | | | **Resultados Esperados** | **Resultados Reales** |
| **CP** | **escenario** | **nombre variable de entrada** | | **valor** |
| 1 report1 | | date | | 2020-05-07 | Devuelve una lista de los intentos de autenticación realizados en la fecha 2020-05-07. | Devuelve una lista de los intentos de autenticación realizados en la fecha 2020-05-07. |
| 2 | report1 | date | | 2020-02-08 | Devuelve null ya que en esta fecha no se realizaron intentos de autenticación | Devuelve null ya que en esta fecha no se realizaron intentos de autenticación |

Tabla 10 Caso de prueba report2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Historia de Usuario** | | | Report2 | | | |
| **Desarrollador** | | | Estrella Cárdenas | | | |
| **Probador** | | | Estrella Cárdenas | | | |
| **Fecha** | | | **29/10/2019** | | | |
| **Versión** | | | **1.0** | | | |
| **Objetivo de la Prueba** | | | Comprobar que funcione correctamente | | | |
| **Descripción de la prueba** | | | Resumen de los intentos de autenticación de un usuario según el tipo por mes. | | | |
| **Condiciones** | | | Existir autenticaciones en la base de datos | | | |
| **Combinaciones de valores de entrada** | | | | | **Resultados Esperados** | **Resultados Reales** |
| **CP** | **escenario** | **nombre variable de entrada** | | **valor** |
| 1 report2 | | username | | star | Devuelve una lista de los intentos de autenticación del usuario star | Devuelve una lista de los intentos de autenticación del usuario star |
| 2 | report2 | username | | alexa | Devuelve null ya que el usuario alexa no está en la base de datos | Devuelve null ya que el usuario alexa no está en la base de datos |

Tabla 11 Caso de prueba report3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Historia de Usuario** | | | Report3 | | | |
| **Desarrollador** | | | Estrella Cárdenas | | | |
| **Probador** | | | Estrella Cárdenas | | | |
| **Fecha** | | | **29/10/2019** | | | |
| **Versión** | | | **1.0** | | | |
| **Objetivo de la Prueba** | | | Comprobar que funcione correctamente | | | |
| **Descripción de la prueba** | | | Resumen de los intentos de autenticación ilegítimas de ese usuario. | | | |
| **Condiciones** | | | Existir autenticaciones en la base de datos | | | |
| **Combinaciones de valores de entrada** | | | | | **Resultados Esperados** | **Resultados Reales** |
| **CP** | **escenario** | **nombre variable de entrada** | | **valor** |
| 1 report3 | | username | | star | Devuelve una lista de intentos de autenticación ilegítimos del usuario star | Devuelve una lista de intentos de autenticación ilegítimos del usuario star |
| 2 | report3 | username | | pepe | Devuelve null ya que el usuario pepe no está en la base de datos | Devuelve null ya que el usuario pepe no está en la base de datos |

Tabla 12 Caso de prueba report4

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Historia de Usuario** | | | Report4 | | | |
| **Desarrollador** | | | Estrella Cárdenas | | | |
| **Probador** | | | Estrella Cárdenas | | | |
| **Fecha** | | | **29/10/2019** | | | |
| **Versión** | | | **1.0** | | | |
| **Objetivo de la Prueba** | | | Comprobar que funcione correctamente | | | |
| **Descripción de la prueba** | | | Resumen de las imágenes usadas en los intentos de autenticación clasificadas dudosas legítimas ordenadas por la frecuencia de uso de un usuario dado. | | | |
| **Condiciones** | | | Existir autenticaciones en la base de datos | | | |
| **Combinaciones de valores de entrada** | | | | | **Resultados Esperados** | **Resultados Reales** |
| **CP** | **escenario** | **nombre variable de entrada** | | **valor** |
| 1 report4 | | username | | star | Devuelve un listado de las imágenes usadas en los intentos de autenticación dudosos legitimo del usuario star | Devuelve un listado de las imágenes usadas en los intentos de autenticación dudosos legitimo del usuario star |
| 2 | Report4 | username | | pipo | Devuelve null ya que el usuario pipo no está en la base de datos | Devuelve null ya que el usuario pipo no está en la base de datos |

Tabla 13 Caso de prueba report5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Historia de Usuario** | | | Report5 | | | |
| **Desarrollador** | | | Estrella Cárdenas | | | |
| **Probador** | | | Estrella Cárdenas | | | |
| **Fecha** | | | **29/10/2019** | | | |
| **Versión** | | | **1.0** | | | |
| **Objetivo de la Prueba** | | | Comprobar que funcione correctamente | | | |
| **Descripción de la prueba** | | | Resumen de los intentos de autenticación en un rango de fecha dado | | | |
| **Condiciones** | | | Existir autenticaciones en la base de datos | | | |
| **Combinaciones de valores de entrada** | | | | | **Resultados Esperados** | **Resultados Reales** |
| **CP** | **escenario** | **nombre variable de entrada** | | **valor** |
| 1 report5 | | dateIni =  dateFin = | | 2020-11-28  2021-10-18 | Devuelve un listado de todos los intentos de autenticación realizados en el rango de fecha 2020-11-28 al 2021-10-18 | Devuelve un listado de todos los intentos de autenticación realizados en el rango de fecha 2020-11-28 al 2021-10-18 |
| 2 | report5 | dateIni =  dateFin = | | 2021-10-20  2021-10-28 | Devuelve null ya entre 2021-10-20 y 2021-10-28 no existen ningún intento de autenticación realizados | Devuelve null ya entre 2021-10-20 y 2021-10-28 no existen ningún intento de autenticación realizados |

A continuación se muestra las pruebas de caja blanca realizadas

Tabla 14 Caso de prueba de caja blanca

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Desarrollador** | | | Estrella Cárdenas | | | |
| **Probador** | | | Estrella Cárdenas | | | |
| **Fecha** | | | 29/10/2019 | | | |
| **Versión** | | | 1.0 | | | |
| **Objetivo de la Prueba** | | | Probar que los métodos funcionen correctamente | | | |
| **Combinaciones de valores de entrada** | | | | | **Resultados Esperados** | **Resultados Reales** |
| **CP** | **escenario** | **nombre variable de entrada** | | **valor** |
| 1 | createPicture | Date  Id  Name  Quality  url | | 2021-10-29  1  o.jpg  alta  images/o-6061015493021598988.png | Devuelve un String= Picture Created | Devuelve un String= Picture Created |
| 2 | deletePicture | Id | | 23 | Devuelve un String= Picture Deleted | Devuelve un String= Picture Deleted |
| 3 | deletePicture | id | | 30 | Devuelve un null ya que no existe el id=30 en la base de datos | Devuelve un null ya que no existe el id=30 en la base de datos |
| 4 | updatePicture | Date  Id  Name  Quality  url | | 2021-10-29  1  a.jpg  alta  images/o-6061015493021598988.png | Devuelve un String= Picture Update | Devuelve un String= Picture Update |
| 5 | createUser | Userid  Identification  Gender  Profid  Username  Fullname  Registered  Roleid  Email | | 34  04041655367  Masculino  12  Alexander  Alexander  0  2  alexandermedero@gmail.com | Devuelve un String= User Created | Devuelve un String= User Created |
| 6 | deleteUser | id | | 3 | Devuelve un String= User Deleted | Devuelve un String= User Deleted |
| 7 | deleteUser | id | | 100 | Devuelve un null ya que no existe el id=100 en la base de datos | Devuelve un null ya que no existe el id=100 en la base de datos |
| 8 | updateUser | Userid  Identification  Gender  Profid  Username  Fullname  Registered  Roleid  Email | | 1  00060566498  femenino  1  star  estrella  1  1  estrellacardenasblanco@gmail.com | Devuelve un String= User Update | Devuelve un String= User Update |

# Referencias

1. Legón, C.M., et al., *Nuevo modelo probabilístico en autenticación gráfica.* Ingeniería Electrónica, Automática y Comunicaciones, 2019. **40**(3): p. 92-104.

2. Johnson, R., et al., *The spring framework–reference documentation.* interface, 2004. **21**: p. 27.

3. Webb, P., et al., *Spring boot reference guide.* Part IV. Spring Boot features, 2013. **24**.

4. Reddy, K.S.P., *PrimeFaces Beginner's Guide*2013: Packt Publishing Ltd.

5. PostgreSQL, D., *Disponível em:* [*http://www*](http://www)*. postgresql. org.* Acesso em, 2005. **8**.

6. Banerjee, U. and E. Narasimhan, *Generating test cases for functional testing of a software application*, 2014, Google Patents.