UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR

POSGRADOS

MAESTRÍA EN

SOFTWARE CON MENCIÓN EN DESARROLLO WEB Y MÓVIL

RPC-SO-34-NO.778-2021

OPCIÓN DE TITULACIÓN:

PROYECTO DE TITULACIÓN CON COMPONENTES DE INVESTIGACIÓN APLICADA Y/O DE DESARROLLO

TEMA:

REDISEÑO Y MEJORA DE LA ARQUITECTURA DE LA PLATAFORMA INFORMÁTICA DEL PROYECTO "PHISHING DE SEGURIDAD COGNITIVA"

AUTOR(ES)

MANUEL VLADIMIR GÓMEZ PLACENCIA

DIRECTOR:

RODRIGO EFRAÍN TUFIÑO CÁRDENAS

QUITO – ECUADOR

2023

Autor(es):

Manuel Vladimir Gómez Placencia
Ingeniero en Sistemas y Computación
Candidato a Magíster en Software por la Universidad Politécnica Salesiana – Sede Quito.
mgomezp8@est.ups.edu.ec

Dirigido por:

Rodrigo Efraín Tufiño Cárdenas
Ingeniero de Sistemas
Master Universitario en Ciencias y Tecnologías de la Computación.
Master Universitario en Software Libre
rtufino@ups.edu.ec

Manuel Vladimir Gómez Placencia
REDISEÑO Y MEJORA DE LA ARQUITECTURA DE LA PLATAFORMA INFORMÁTICA DEL PROYECTO "PHISHING DE SEGURIDAD COGNITIVA"

DEDICATORIA

A mi madre, Norma Placencia Galindo, quien siempre me impulsó a ser mejor y confió en mis capacidades a pesar de toda adversidad y me animó a superar retos en todo aspecto de mi vida con su inagotable amor y comprensión. Donde quiera que estés, te extraño mucho y te agradezco aún mucho más.

A mi padre, Manuel Gómez De Labastida, por su constancia para lograr que salgamos adelante y que todas nuestras metas se cumplan.

A mi querido hermano, Cesar, por su cariño e inspiración para siempre perseverar y no darse por vencido.

A mis amigos y compañeros, quienes han estado siempre pendientes de este logro y me han brindado su apoyo y soporte en incontables ocasiones.

Para todos ellos, dedico este trabajo.

AGRADECIMIENTO

Quisiera expresar mi agradecimiento a todas las personas que fueron parte fundamental para la realización de este trabajo de maestría. En primer lugar, quiero agradecer a mi tutor Rodrigo Tufiño por su orientación, paciencia y apoyo constante a lo largo de este proceso. Sus conocimientos y consejos fueron fundamentales para llevar a cabo esta investigación.

A mi familia, les agradezco por su inquebrantable apoyo y por ser mi fuente de inspiración y motivación. Su amor y aliento han sido un motor fundamental para alcanzar esta meta.

A mis colegas de trabajo, agradezco su colaboración, comprensión y la flexibilidad que me brindaron para equilibrar mis responsabilidades laborales con mis estudios, así como sus ideas y retroalimentación en la construcción de la solución informática.

En resumen, este logro no habría sido posible sin el apoyo de todas estas personas. A todos los mencionados, mis más sinceros agradecimientos.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Rediseño y mejora de la arquitectura de la plataforma informática del proyecto "Phishing de Seguridad Cognitiva"

Autor(es):

Manuel Vladimir Gómez Placencia

Octubre 2023

Resumen

El presente proyecto es un estudio enfocado en la refactorización y mejora de una aplicación creada para la el entrenamiento en detección de phishing. Como parte de un estudio de seguridad cognitiva, se construyó un prototipo de aplicación web orientado a evaluar la capacidad de las personas para distinguir mensajes de email fraudulentos y medir su percepción general de seguridad informática con el cual se obtuvo información valiosa para el estudio. Sin embargo se reportaron varios problemas con la plataforma debido principalmente a la falta de mantenimiento adecuado desde su implementación por lo que se decidió realizar un trabajo para mejorar la calidad del sistema.

Se planteó una metodología sencilla para guiar nuevos proyectos de mantenimiento de software utilizando como base los requerimientos de usuario y el análisis contextual de cada componente y su funcionamiento interno, conjuntamente con la ayuda de métricas y herramientas de calidad de software de uso general. Luego de este análisis, se plantearon varias soluciones para los defectos observados y se estimó el esfuerzo de mitigación comparado con el esfuerzo de rediseñar del sistema, partiendo desde su arquitectura.

Teniendo en cuenta estos factores, se consideró factible una reconstrucción completa del sistema. Se utilizó un proceso de desarrollo ágil, integrando el mantenimiento como una de las principales tareas para mantener un mejor control sobre defectos y mejoras. Finalmente, se creó una nueva aplicación que brinda funcionalidad mejorada y se estima que podrá tener un mejor perfil de mantenimiento en el futuro.

En este documento se detallan las varias etapas de todo el proceso, decisiones tomadas, particularidades y resultados obtenidos.

Palabras clave:

Arquitectura de software, refactorización, phishing, seguridad cognitiva.

Abstract

This project is a study focused on the refactoring and improvement of an application made for phishing detection training. As part of a cognitive security study, a prototype web application was developed in order to assess people's ability to distinguish fraudulent email messages and measure their overall perception of computer security. Valuable information was obtained for the study using this system; however, several issues with the platform were reported, due mainly to the lack of proper maintenance since its implementation, so it was decided to carry out work to improve the overall quality of the system.

A simple methodology was proposed to guide new software maintenance projects based on user requirements and contextual analysis of each component and its inner workings, together with the help of general-purpose software quality metrics and tools. After this analysis, several solutions were proposed to solve the observed defects and the mitigation effort was estimated compared to the effort of a system redesign, starting from its architecture.

Considering these factors, a full system rebuild was deemed feasible. An agile development process was used, integrating maintenance as one of the main tasks to keep better control over defects and improvements. Finally, a new application was built with improved functionality and, it is estimated will have a better maintenance profile in the future.

This document details the various stages of the entire process, decisions made, specific aspects and obtained results.

Keywords:

Software architecture, refactoring, phishing, cognitive security.

1. Introducción

La masificación del uso de internet en una gran mayoría de los aspectos de la vida moderna ha significado un incremento en el número y variedad de ataques informáticos dirigidos a vulnerar los datos de personas y empresas lo que requiere de una constante mejora de los mecanismos de seguridad en los sistemas informáticos actuales.

Uno de los ataques informáticos más populares en la actualidad es el "phishing", el cual está catalogado como un ataque de ingeniería social que se basa en engañar a los usuarios presentando mensajes de email, sitios web, mensajes SMS y otras formas de comunicación que tienen por objetivo suplantar la identidad de entidades y personas con el fin de obtener información privada o distribuir malware (Sumner & Yuan, 2019) (Vijayalakshmi et al., 2020).

La mayoría de las soluciones actuales para detectar ataques de phishing son automáticas y muchas de ellas se basan en técnicas de Machine Learning entrenadas para detectar si un mensaje proviene de una fuente auténtica o no (Divakaran & Oest, 2022) (Vijayalakshmi et al., 2020). Estas herramientas son la primera línea de defensa y filtran una buena cantidad de mensajes fraudulentos protegiendo a los usuarios de forma transparente. Sin embargo, no son perfectas, el grado de sofisticación y sutileza de los ataques se incrementa constantemente por lo cual es necesario que los mismos usuarios estén en capacidad de protegerse cuando los sistemas automáticos fallen (Sumner & Yuan, 2019).

Para lograr esto, se requiere que la gente posea un cierto nivel de entrenamiento que les permita reconocer mensajes fraudulentos y puedan actuar para protegerse. Una forma de conocer cuál es la capacidad de una persona para reconocer mensajes fraudulentos es utilizando test sencillos los cuales pueden automatizarse y ponerse a disposición del público a través de tecnologías masificadas actualmente, como aplicaciones web y aplicaciones móviles (Al-hamar & Kolivand, 2020).

Como parte del proyecto de investigación de la Universidad de las Fuerzas Armadas ESPE denominado “Detección y mitigación de ataques de Ingeniería Social utilizando Seguridad Cognitiva”, se desarrolló una aplicación web prototipo que permite evaluar a los usuarios en su capacidad de reconocer mensajes fraudulentos (Briceño et al., 2021).

La aplicación informática original del proyecto ha sido utilizada en escenarios reales con usuarios de instituciones públicas y privadas quienes han sido evaluados para determinar su capacidad para distinguir mensajes electrónicos fraudulentos y mitigar este riesgo. Los resultados de estas evaluaciones demuestran que la funcionalidad de este sistema puede utilizarse de forma más amplia lo cual beneficiaría a muchas más personas con el entrenamiento para defenderse contra ataques y engaños basados en información fraudulenta. Adicionalmente, el uso de esta herramienta proporcionará más datos para proyectos de investigación en el campo de seguridad cognitiva (Briceño et al., 2021) (Sutter et al., 2022).

Por este motivo, se propone mejorar la plataforma para que tenga una mayor capacidad y funcionalidad con el fin de incrementar su utilidad en la investigación de técnicas de seguridad cognitiva contra ataques de phishing.

2. Marco Teórico Referencial

El phishing es un tipo de ataque informático de ingeniería social que se basa en engañar a los usuarios presentando mensajes que suplantan la identidad de entidades y personas con el fin de obtener información privada o distribuir malware (Sutter et al., 2022) (Divakaran & Oest, 2022).

Dentro de las técnicas actuales para detectar ataques de phishing, las más utilizadas se basan en aplicaciones de Machine Learning (Vijayalakshmi et al., 2020) (Divakaran & Oest, 2022) que son entrenadas para detectar varios aspectos de los mensajes electrónicos y clasificarlos como amenaza o no. Los dos aspectos más comunes que se analizan en los mensajes son los hipervínculos que contienen y el contenido del mensaje en sí.

A pesar de los avances en la automatización de la detección de phishing, la sofisticación de los ataques se incrementa y refina cada día por lo que se recomienda que los usuarios sean concientizados sobre problema y de las formas de mitigarlo por ellos mismos en combinación con las herramientas automáticas (Sumner & Yuan, 2019) (Sutter et al., 2022).

Estudios recientes muestran que usuarios con alta formación académica e incluso expertos en el uso de tecnología pueden ser susceptibles a caer en engaños de phishing (Diaz et al., 2018) (Wash, 2020) por lo que se recomienda disponer de recursos de entrenamiento cognitivo para que la gente se informe y conozca más sobre este problema (Sutter et al., 2022) (Desolda et al., 2022) (Zhuo et al., 2022). En este sentido, se han publicado varias soluciones comerciales de entrenamiento en detección de phishing como https://caniphish.com, https://www.phishingbox.com y https://phishinsight.trendmicro.com que permiten evaluar a los participantes utilizando simulaciones de mensajes de phishing en forma de test. Iniciativas similares se han generado también desde el sector académico como parte de estudios más profundos de seguridad cognitiva (Sutter et al., 2022) (Briceño et al., 2021) (Al-hamar & Kolivand, 2020).

La plataforma de entrenamiento de phishing objeto del presente estudio es una de estas soluciones y ha sido utilizada para la evaluación de personas con efectividad (Briceño et al., 2021), sin embargo se han detectado varias falencias técnicas y de funcionalidad en el sistema por lo que, para garantizar el funcionamiento a futuro de esta aplicación, es necesario realizar cambios dentro de un proceso de refactorización (Fowler, 2018) que tiene como propósito mejorar la calidad del software.

El proceso de determinar si un sistema debe ser refactorizado se basa en el reconocimiento de ciertas características en el software que pueden indicar problemas más profundos (Fowler, 2018) (Kerievsky, 2004). Estas características sospechosas se conocen como "code smells" y existe una amplia documentación y taxonomía de los más comunes y formas de solucionarlos aplicando técnicas simples y patrones de diseño (Chathuranga, 2021) (Kaya et al., 2018) (Kerievsky, 2004) (Alkhaeir & Walter, 2021). De la misma forma, actualmente existen una gran variedad de herramientas automáticas de análisis estático de código que permiten detectar algunas porciones de código que son comúnmente problemáticas y recomendar mejores prácticas (Kim et al., 2012) (Kaur et al., 2021) (Mohanani et al., 2022). Muchas de estas herramientas vienen incorporadas en IDEs modernas como Visual Studio y la gama de herramientas de Jetbrains para varios lenguajes.

La amplia disponibilidad de estas herramientas hace que aplicar refactorizaciones comunes sea un proceso simple y directo, pero varios estudios han encontrado que para realmente mejorar la calidad del software es necesario considerar el contexto de donde se encuentra el problema (Sae-Lim et al., 2018) (Golubev et al., 2021). La base de estos estudios es el feedback directo proporcionado por equipos de desarrollo, sus experiencias y percepción del proceso.

Adicionalmente, se encontró que una buena planificación para proyectos de mejora de software debe considerar otros factores como una estimación del esfuerzo requerido, los potenciales riesgos de los cambios y la deuda técnica en la que se puede incurrir (Lacerda et al., 2020) (Martini et al., 2018) (Ó Cinnéide et al., 2016) (Peruma et al., 2022). Una conclusión recurrente en los estudios es que es necesario involucrar cierto nivel de revisión manual del código, lo cual permite

entender el propósito del código afectado más allá de estructuras o sentencias específicas detectadas como code smells (Pascarella et al., 2019) (Golubev et al., 2021) (Han et al., 2021). Los estudios demuestran evidencias de situaciones donde una refactorización podría impactar negativamente al sistema en algún aspecto no considerado solo por un código poco limpio (Sae-Lim et al., 2018) (Traini et al., 2021). A continuación, se revisa en más detalle el proceso de refactorización.

2.1. Proceso de Refactorización

En la mayoría de la literatura clásica que trata sobre el tema de refactorización, se describe el proceso práctico de como cambiar la estructura interna del código del sistema para mejorar su mantenibilidad pero no cambiar su funcionalidad, sin embargo en pocos trabajos se describe cómo empezar con el proceso de una forma metodológica (Suryanarayana et al., 2014). Muchos de los trabajos sobre refactorización consisten en un catálogo de los problemas, como detectarlos y una solución propuesta. Los principales problemas del código que son sujetos a refactorización se los conoce como “code smells”, que podría traducirse como código sucio; una programación que si bien funciona, puede presentar falencias como falta de claridad, ineficiencia, demasiada complejidad, alto acoplamiento, baja cohesión entre otros defectos que disminuyen la calidad general del sistema (Kerievsky, 2004) (Fowler, 2018).

Varios autores como Martin Fowler en su libro Refactoring (Fowler, 2018), proponen que el requerimiento inicial para empezar con las refactorizaciones al detectar code smells es contar con pruebas unitarias de la funcionalidad que permitan realizar los cambios e ir verificando que el resultado siga siendo el esperado. Esto no suele ser posible en todos los casos, incluso en proyectos donde se han seguido buenas prácticas como el uso extendido de pruebas unitarias ya que no toda la funcionalidad puede estar cubierta. Si se utiliza un criterio de priorización de características, muchos tests estarán orientados a aspectos críticos del sistema y no a cada punto de función, clase o bloque de código existente, es decir, se habrán creado pruebas unitarias de acuerdo al contexto de la funcionalidad y no solo a buscar un mayor porcentaje de cobertura.

En otros casos, simplemente no existen pruebas unitarias por lo que es necesario tomar otras consideraciones para decidir cómo abordar el proceso de mejora del software. En (Suryanarayana et al., 2014) se plantea que el proceso de refactorización debe iniciar con una revisión manual del código que permita identificar los puntos donde se deberá analizar el código de forma más minuciosa. La extensión del código a revisar se determina por el alcance del proyecto de refactorización, pudiendo abarcar todo el proyecto o solo ciertos módulos en particular.

En el caso de la arquitectura del sistema, existen métodos que permiten evaluar aspectos de diseño y complejidad con enfoque en minimizar riesgos y asegurar una buena mantenibilidad del software (Samarthyam et al., 2016) (Briceño et al., 2021). Varios de estos métodos (como ATAM) se basan en la presentación formal de requerimientos de negocio y varios escenarios de arquitecturas posibles que serán evaluados con los stakeholders del proyecto en pasos definidos, por lo que se recomienda realizar este proceso al inicio del proyecto cuando no se tiene una gran base de código establecida. A pesar de esto, es posible utilizar las partes y métricas de evaluación dinámica de arquitectura de estos métodos para un proyecto de refactorización donde se considera una mejora a nivel de diseño, eso sí, teniendo en cuenta los posibles riesgos y costos de este enfoque.

Habiendo definido qué se va a revisar, es necesario determinar cómo se va a revisar el código y los insumos técnicos del sistema. Una revisión manual de cada parte del sistema es posible, pero esto puede llegar a ser sumamente laborioso y complejo dependiendo del sistema en cuestión, por lo que se recomienda complementar el proceso con software de análisis especializado.

Desde que la refactorización se estableció como una buena práctica para mejorar la calidad del software, la industria y la comunidad de desarrollo han creado herramientas automatizadas que permiten detectar defectos en el código fuente a partir de catálogos de varios tipos de anomalías comunes. Estas herramientas se las conoce como software de análisis estático (Sobhy et al., 2021). Algunos de los defectos que se pueden detectar son errores de uso del lenguaje de programación, fallas de seguridad conocidas, malas prácticas, anti-patrones y “code smells”,

posibles problemas de complejidad ciclomática, entre otros (Chathuranga, 2021) (Kerievsky, 2004) (Fowler, 2018) (Suryanarayana et al., 2014).

Si bien estas herramientas ayudan a detectar varios defectos en el código, muchas veces es necesario configurar el nivel de detección ya que, por lo general, no se considera el contexto de aplicación del código, es decir para qué se está utilizando y como interactúa con otros componentes o incluso con el usuario directamente. También es necesario considerar las prácticas y convenciones del equipo de desarrollo debido a que, por defecto, los analizadores suelen levantar advertencias sobre cosas como el formateo del código fuente o nombres de variables que se salen de convenciones comunes y que pueden no ser las adoptadas en el proyecto analizado lo que deriva en la creación de reglas personalizadas.

Refactorización vs resolución de errores

Es necesario mencionar la diferencia entre un proceso de refactorización y uno de resolución de errores detectados. Un proceso de refactorización por lo general es algo planificado que sale de una revisión como es el caso del presente proyecto mientras que la corrección de errores detectados es un proceso más reactivo donde, por lo general, se requiere de una intervención inmediata para corregir la condición adversa.

En muchas ocasiones, la resolución del error también involucra una refactorización la cual puede ser, de hecho, la solución necesaria. Sin embargo, no es usual que los procesos de corrección de errores se ingresen en una planificación de desarrollo basada en Scrum u otra metodología aunque existen casos donde se detectan errores que solo se dan en condiciones muy específicas o que no afectan directamente a la consistencia de datos o flujos importantes del sistema. Para este caso, si es posible ingresar tareas de corrección de errores dentro de la planificación del proyecto, por ejemplo en el backlog del sprint actual o en los siguientes sprints.

2.2. Metodología

El sistema de detección de phishing, objeto de análisis de este proyecto, es un prototipo funcional que no contiene una extensa funcionalidad y no maneja un volumen grande de datos por lo que se determinó que es factible realizar una revisión general que incluye todos los componentes programados cuyo código esté disponible.

Se propone el siguiente el proceso para identificar defectos y oportunidades de mejora:

- Análisis de Requerimientos e insumos.
- Revisión general: código fuente, documentación, bases de datos.
- Identificación de tecnologías y herramientas para el análisis.
- Revisión del diseño, arquitectura y organización del sistema.
- Análisis estático del código.
- Conclusiones del análisis.
- Propuesta de rediseño.

Inicialmente, se identificó qué insumos técnicos y de documentación se encuentran disponibles para realizar una revisión. Se determinó que al existir un documento de diseño del sistema que incluye un detalle de requerimientos, es posible realizar un análisis refinado de estos requerimientos que incluya las ideas de los stakeholders y del autor para incorporar mejoras a la plataforma.

Dentro del análisis técnico, se realizó inicialmente una revisión manual del sistema considerando el contexto de cada componente con el fin de determinar cómo está hecho el sistema, qué herramientas se utilizan y cómo orientar el resto del análisis de acuerdo a la estructura general observada. Como complemento, se hicieron pruebas con herramientas automáticas para identificar detalles cuantificables de la base de código, así como defectos y posibles fallas. No se incluyó el código de bibliotecas externas más allá de verificaciones de obsolescencia y chequeos de seguridad.

Para la revisión de arquitectura y diseño se usó el criterio profesional del autor además de las definiciones del documento de diseño original solo como comparativa, orientando el análisis hacia un punto de vista funcional y de mantenibilidad con un enfoque en dependencias, despliegue y administración.

Se encontraron problemas más profundos que afectan al sistema en aspectos como mantenibilidad, crecimiento y funcionalidad. La solución de estos problemas es posible utilizando un proceso de desarrollo donde se consideren los requerimientos iniciales (Briceño et al., 2021) (Hussain & Javed, 2015) (Kasauli et al., 2021) así como propuestas de mejora detallados más adelante.

Finalmente, se definieron y planificaron cambios específicos para corregir los defectos y las nuevas funcionalidades propuestas para mejorar la calidad del sistema.

Proceso de desarrollo

Actualmente, el proceso de desarrollo más utilizado para proyectos de software se basa en metodologías ágiles como Scrum y variantes de esta (Ashraf & Shabib, 2017) (Hussain & Javed, 2015). Para este desarrollo se utilizó una metodología ágil iterativa e incremental que considera no solo la solución de problemas existentes encontrados y documentados sino también la reconstrucción de componentes enteros si se determina que el esfuerzo de refactorización no aporta a la mejora general de la calidad del software (Ó Cinnéide et al., 2016) (Hussain & Javed, 2015) usando métricas establecidas.

Dentro de una metodología ágil también se pueden incorporar tareas de refactorización y mejoras las cuales se incluyen en los micro ciclos de desarrollo (sprints), dándoles una prioridad, tiempo y asignación de recursos (Pecorelli et al., 2020) (Hussain & Javed, 2015) (Kasauli et al., 2021). Adicionalmente, se ha encontrado que es posible incorporar tareas específicas orientadas a mejorar o resolver aspectos no funcionales del sistema, como son la experiencia de usuario (Hinderks, 2019), alta disponibilidad (Kasauli et al., 2021), el rendimiento (Traini et al., 2021) y mejoras a la arquitectura base (Wohlfarth & Riebisch, 2006) (Samarthyam et al., 2016).

3. Desarrollo del Proyecto

3.1. Análisis de Requerimientos

Como primer paso del análisis se identificaron las fuentes de información necesarias para evaluar el diseño y construcción de la aplicación. Se encontró que la definición del sistema se encuentra en la tesis de grado “Plataforma Web para entrenamiento de ataques de Phishing mediante seguridad y psicología cognitiva” (Briceño et al., 2021) en donde se encuentra el diseño inicial de la solución así como la descripción de las principales herramientas y tecnologías utilizadas. No se recibieron documentos adicionales ni manuales de usuario del sistema.

Contexto de análisis

Para determinar qué aspectos y funcionalidades de la aplicación deben ser mejoradas o incorporadas, es necesario definir un contexto general que servirá de guía para encaminar el análisis del código y generar contextos específicos centrados en mejorar la calidad del sistema. Este contexto general fue definido por el autor con el fin de dar una dirección a las propuestas de mejora, así como a la solución de errores, considerando que los mejores resultados se suelen dar al conocer los distintos matices tanto técnicos como de negocio del componente, módulo o fragmento de código en el que se está trabajando, según estudios realizados al respecto (Sae-Lim et al., 2018). Estos contextos están ligados en su mayoría a los requerimientos funcionales y no funcionales.

Sobre el tema de mejoras, el concepto de calidad de software es algo relativo al punto de vista de los usuarios y desarrolladores de la aplicación, por lo que los criterios suelen ser variados y subjetivos. En este caso es necesario contrastar los requerimientos planteados por los usuarios contra las posibilidades de implementación técnica con el fin de lograr un adecuado balance entre experiencia de uso versus mantenibilidad (Ó Cinnéide et al., 2016).

El objetivo general de este proyecto planteado por los stakeholders y el autor es: Mejorar la funcionalidad del software de entrenamiento de phishing para que sea más versátil, robusto y amigable, así como obtener datos más útiles sobre los resultados (Briceño et al., 2021).

La parte de esta narrativa que se refiere a características no funcionales como ser “amigable” y “robusto” forma parte de la usabilidad del sistema y de la revisión del código, respectivamente. El aspecto de versatilidad y utilidad de datos obtenidos se refieren a ideas de mejoras las cuales se encuentran en la sección de requerimientos.

3.2. Resumen de Requerimientos

Como primer paso, se extrajeron los requerimientos originales de la plataforma descritos en (Briceño et al., 2021) complementados con nuevas funcionalidades propuestas planteadas por el autor y stakeholders. Estas definiciones fueron contrastadas contra la implementación actual utilizando revisiones manuales de funcionalidad y se generó una tabla del estado de implementación (cumplimiento) por cada caso de uso refinado que se encuentra al final de la sección de requerimientos.

Formato de presentación

Para la presentación de los requerimientos del sistema se utilizó un formato libre basado en historias de usuario lo cual permite combinar una definición formal de la funcionalidad requerida y una descripción más abierta de procesos, interfaces y comportamientos esperados de la aplicación sin dejar a un lado la clasificación y orden de los requerimientos. Un reciente estudio (Mohanani et al., 2022) encontró que el uso forzado de plantillas y formatos estándar para requerimientos tiende a disminuir la creatividad al momento de plasmar las ideas para un producto lo que puede llegar a limitar el potencial de una aplicación a la larga. Esto no quiere decir que el planteamiento de requerimientos deba ser un proceso sin estructura u orden lo cual afectaría directamente al proceso de desarrollo, introduciendo complejidad y confusión innecesaria.

Con esto en mente, para este proyecto se utilizó parte de la estructura general del formato de Especificación de Requerimientos de la IEEE (IEEE, 2018) como se muestra a continuación. Se trata de un documento organizado por secciones que definen el propósito del sistema, sus características y funcionalidades detalladas en casos de uso a partir del documento inicial, expandidos de acuerdo a las necesidades solicitadas por el cliente del sistema.

1. Propósito

El sistema es un portal web donde los usuarios pueden tomar pruebas sobre phishing que les permite determinar su conocimiento y vulnerabilidad ante ataques de este tipo. Los usuarios responden a una prueba con preguntas específicas y al final se devuelve un resultado sobre su riesgo ante phishing. Los usuarios deben ingresar datos básicos sobre su perfil personal y profesional antes de cada prueba. Los resultados se almacenan junto con este perfil para análisis e investigaciones sobre seguridad cognitiva.

1.1. Audiencia

El sistema está dirigido a personas de todo ámbito social y profesional que tengan interés en conocer el riesgo de los ataques de phishing y entrenarse para poder defenderse contra estos. Los datos recopilados pueden servir a la comunidad de investigación de seguridad informática y cognitiva.

1.2. Alcance

Facilitar a los usuarios participantes el acceso a pruebas de phishing que evaluarán su conocimiento y riesgo potencial. Debe ser accesible a través de internet de forma similar a otros servicios que existen actualmente en el mercado.

2. Descripción general

A continuación, se describe el sistema en términos de funcionalidad esperada y características no funcionales, desde un campo general hasta detalles específicos.

2.1. Perspectiva del producto

El sistema es una aplicación web que provee un mecanismo simple para entrenar a personas en detección de phishing. A grandes rasgos el sistema debe permitir:

- Evaluar a usuarios sobre su conocimiento de phishing usando cuestionarios dirigidos.
- Permitir identificar el perfil de los usuarios participantes.
- Obtener resultados de las evaluaciones para presentar al usuario y para posterior análisis en forma de reportes.
- Administrar y mantener la plataforma.

2.2. Características de usuario y casos de uso

Originalmente se definieron 5 perfiles de usuario que eran una mezcla entre un perfil profesional y su función en los procesos del sistema. En un ámbito más general, se considera una mejor categorización usando solo el rol dentro del sistema, siendo así:

2.2.1. Perfiles de usuario

1. *Usuario Evaluado*. Persona que participa en las evaluaciones anti-phishing respondiendo los cuestionarios y revisando sus propios resultados. Esta persona puede acceder al sistema sin necesidad de una cuenta registrada y realizar las evaluaciones.
2. *Editor de contenido*. Persona que revisa, crea y edita las preguntas de los cuestionarios así como configurar los parámetros de evaluación. Se encarga de la información que será vista por los usuarios evaluados. Puede también ser un evaluador.
3. *Consumidor de resultados*. Persona que extrae, analiza y procesa los resultados de las evaluaciones aplicando criterios técnicos. Es el consumidor principal de la reportería del sistema.
4. *Administrador*. Se encarga de que el conjunto del sistema funcione correctamente, incluyendo su operatividad y mantenimiento. Incluye la gestión de accesos al sistema, su seguridad y puede llegar a tareas técnicas de mejoras y despliegue.

2.2.2. Procesos generales

En esta sección se listan los procesos generales que ejecutarán los usuarios y/o el sistema. Para este proyecto se decidió que las tareas específicas o detalles minuciosos de cada proceso serán definidos en las etapas de implementación (sprints), tomando los casos de uso como guías para obtener el resultado deseado. Se han tomado los procesos originales como referencia para obtener esta nueva especificación y se proponen alternativas para ser analizadas posteriormente.

Proceso general	**Capacidades**
Registro de usuario	El usuario podrá ingresar información básica de su perfil personal y profesional para una evaluación o creación de cuenta de usuario persistente.
Ingreso al cuestionario	El usuario llena información de identificación y accede a los cuestionarios de evaluación configurados.
Ejecución de evaluación	El usuario evaluado podrá responder a las preguntas del cuestionario, ingresar información adicional y recibir un resultado de su evaluación al término del proceso.
Creación de contenido	El usuario autorizado podrá crear preguntas, crear cuestionarios y editar el contenido de estos a través de una interfaz web.
Visualización de reportes	Permite visualizar en pantalla información relacionada con las evaluaciones, utilizando filtros definidos. Incluye la exportación de datos en formato tabular en ciertos reportes.
Administración del sistema	El usuario administrador podrá gestionar la configuración del aplicativo, cambiar parámetros de funcionamiento y editar catálogos de datos. Se podrá también monitorear el uso y salud del sistema a través de registros de auditoría y otra información técnica. Adicionalmente, se administrarán los accesos de los usuarios al sistema, credenciales y permisos por perfil.
Cálculos de evaluación	El sistema posee algoritmos y fórmulas para calcular el nivel de detección de phishing a partir de los datos de evaluaciones registradas así como métodos para seleccionar preguntas con base en respuestas y tiempos anteriores en la evaluación.

Tabla 1. Procesos generales del sistema

2.2.3. Casos de uso

Para la definición de los casos de uso, se tomaron los requerimientos funcionales originales (Briceño et al., 2021) y se han replanteado como historias de usuario sin restricción de formato, en donde se han incluido ciertos detalles de implementación o características deseadas, conforme a la definición del formato para este proyecto. Adicionalmente, se incluyó el estatus de implementación de cada caso, con el fin de usarlo en la planificación de las tareas de desarrollo posteriores al análisis inicial.

CU01: Iniciar evaluación de cuestionario
El usuario autorizado ha navegado hasta la página de inicio del cuestionario. Se muestra una pantalla de introducción la cual muestra información de cómo responder las preguntas y utilizar el sistema. Se presenta un botón de Iniciar el cual comenzará la evaluación, el sistema registrará esta acción y se creará un registro que incluye el estado del proceso.
Características y restricciones: El usuario deberá haber ingresado a la pantalla de registro para evaluación y completado los datos del formulario inicial. El sistema deberá registrar el tiempo de respuesta de cada pregunta y el total del cuestionario. El sistema deberá contener ayudas en pantalla para el usuario. Se deberá guardar el avance del usuario en sus respuestas. El cuestionario debe tener un estado general el cual deberá ser persistente en casos de falla de red o si el usuario sale del sitio de forma inesperada. El sistema determinará el orden y tipo de las preguntas a presentar (configuración) Referencias: RF01, RF04, RF06
Otras ideas: Se podría permitir avanzar o retroceder en las preguntas del cuestionario para completarlo en desorden si se desea.
Estatus: Implementado

CU02: Responder a una pregunta
Dentro del cuestionario, el usuario visualiza una pregunta sobre phishing la cual contiene un enunciado, el contenido a evaluar y controles para responder. El contenido de la pregunta puede ser: • Una representación de un mensaje de email con texto, imágenes e hipervínculos los cuales pueden ser examinados por el usuario.

• Un cuestionario con preguntas sobre precepción de seguridad informática Los controles permiten responder si el mensaje es o no un intento de phishing o en el caso del cuestionario de precepción, guardar las respuestas y avanzar. Se incluye una caja de texto para comentarios. El usuario hace clic sobre una de las opciones de respuesta y el sistema pide una confirmación, luego de lo cual se guarda la respuesta y datos adicionales y se procede a la siguiente pregunta o al final del cuestionario según el flujo. Referencias: RF06, RF07, RF08, RF09
Características y restricciones: Solo para preguntas de phishing: • El sistema debe permitir ver la dirección a donde apunta un hipervínculo cuando se pasa el cursor del ratón sobre el texto tal como se presenta en situaciones reales. • Se deben registrar los eventos de clic y hover sobre vínculos en cada ejercicio. • Se debe registrar el tiempo que el usuario se demora en responder cada pregunta y el tiempo total del test. (CU01). Se deberá registrar el comentario que el usuario puede dejar opcionalmente al terminar su respuesta.
Estatus: Implementado

CU03: Ver resultados de la evaluación
Al llegar a la última pregunta, se le pide al usuario que confirme que ha terminado la evaluación, puede ser con un popup o con una página final y un botón de confirmación. El usuario hace clic en el botón y el sistema calcula y guarda los resultados. El sistema presenta tres conjuntos de resultados: • Score por cada respuesta, score total. • Tiempo total y promedio por respuesta. • Nivel de detección del usuario con un puntaje de percepción. El usuario podrá ver una explicación sobre cada ejercicio de phishing además de los comentarios ingresados en su propia evaluación. El usuario puede terminar el ejercicio navegando a otra sección o con un botón para cerrar la página. Referencias: RF010, RF011
Características y restricciones: Los resultados deberán ser fáciles de interpretar. Se deberá generar un gráfico del score (implementación). Los cálculos se realizarán internamente de acuerdo con la configuración del test.
Estatus: Implementado

Ideas adicionales: Formatear el contenido para impresión. Envío opcional de correo electrónico con los resultados al usuario si se dispone de la infraestructura necesaria.

CU04: Crear/Editar pregunta
El usuario autorizado ingresa la opción de crear o editar una pregunta. Los datos para configurar una pregunta son: • Contenido a mostrar. • Es phishing: Si o No. • Dificultad de la pregunta, número de 1 a 10. • Explicación de la respuesta. • Categoría/Tipo. • Habilitada: Si o No. El editor de contenido permite: • Editar código HTML. • Cargar imágenes. • Definir hipervínculos como Hotspots. • Determinar posiciones/coordenadas para rastreo de clics. Se tendrá una previsualización de la pregunta ya sea cerca del editor o como otra ventana/pestaña. El usuario puede cambiar el contenido y guardar los cambios. El sistema validará los campos requeridos y formatos de datos y si no pasa la validación se genera una alerta.
Características y restricciones: Se puede dar un o varias categorías a la pregunta, por ejemplo si se refiere a phishing bancario, suplantación de identidad personal, compras, etc. Las preguntas podrán utilizarse en varias pruebas configuradas.
Estatus: NUEVO
Ideas adicionales: Utilizar un editor de código fuente en línea como CodeMirror

CU05: Ingreso de usuario registrado
El usuario navega a la sección para usuarios registrados. El usuario ingresa sus credenciales: correo electrónico y contraseña. El sistema valida los datos. Si no son válidos se muestra una alerta de seguridad. Si los datos son correctos, el sistema muestra la pantalla inicial o “home” del usuario de acuerdo a su perfil.
Características y restricciones:

Todos los intentos de ingreso exitosos o no deben registrarse en auditoría.
Otras ideas: En futuras versiones se podría incorporar la funcionalidad para que usuarios externos generen sus propias cuentas en el sistema y puedan ver un registro de sus resultados así como reportes personalizados. Se podría incorporar seguridad adicional como registro de intentos fallidos y protección por captcha u otro mecanismo para evitar ataques.
Status: NUEVO

CU06: Ingreso a evaluación configurada
El usuario navega a la opción de iniciar test de detección de phishing. El usuario llena datos de perfil (nombres, profesión, edad, experiencia, etc.) El sistema valida la información ingresada, la almacena y se continúa al caso CU01, Iniciar la evaluación.
Características y restricciones: El sistema valida los datos del formulario de registro. Opcionalmente, el sistema puede recuperar información previamente ingresada si el usuario repite el test y llenar parcialmente el formulario.
Estatus: Implementado PARCIALMENTE

CU07: Crear/editar plantilla de evaluación
El usuario autorizado ingresa a la opción de editar evaluación. El usuario ingresa los datos de la plantilla: • Descripción y nombre. • Seleccionar las preguntas del banco existente (habilitadas). • Incluir opcionalmente el cuestionario de percepción y posición dentro de la evaluación. • Habilitada para uso general si/no. El sistema valida la información ingresada y almacena los datos.
Características y restricciones: El sistema deberá mostrar una advertencia si se han seleccionado preguntas deshabilitadas. Se podría incluir un simulador del test para que el editor verifique la coherencia de las preguntas, duración y otros parámetros.
Estatus: NUEVO

CU08: Reportes de evaluación
El usuario autorizado ingresa al módulo de reportes. El usuario selecciona un reporte

En la pantalla del reporte se podrán ver como mínimo los siguientes elementos: • Filtros de datos como fechas, tipos de evaluación, tipos de usuarios, sesiones, etc. • Contenido del reporte: tabla de datos y/o gráfico Opcionalmente, se podrá dar la opción de exportar los datos tabulares a un formato procesable como archivos Microsoft Excel .xlsx.
Características y restricciones: La particularidad de cada reporte se definirá con los stakeholders durante el desarrollo de esta funcionalidad. El acceso a reportes será controlado por permisos de la aplicación
Estatus: NUEVO

CU09: Administración del sistema
El usuario administrador podrá realizar la siguiente funcionalidad • Listar, crear, editar y eliminar usuarios registrados. • Gestionar acceso y credenciales de usuarios. Resetear contraseñas y habilitar acceso. • Administrar perfiles de usuario. Editar, poner y quitar diferentes permisos de acceso a las funcionalidades por cada perfil que será asignado a usuarios registrados. • Definir perfil de acceso para usuarios externos que se registren (opcional). *Auditorías* El administrador podrá revisar el log de eventos del sistema que contará con filtros por tipo de evento, fecha y descripción. Adicionalmente, el administrador podrá realizar las mismas tareas de gestión de datos de los casos de uso que requieren un usuario autorizado (CU04, CU05, CU07) y ver los reportes del caso CU08.
Características y restricciones: Los usuarios que sean registrados tendrán un perfil definido por defecto el cual podría ser cambiado por el administrador. Existen tareas que los usuarios administradores realizan fuera de la interfaz del sistema las cuales dependerán del entorno de despliegue y las políticas de la institución y no se consideran dentro de los requerimientos.
Estatus: NUEVO

2.3. Entorno Operativo

El sistema se compone de los siguientes componentes:

- Una interfaz accesible desde navegador web sin restricción de dispositivo.
- Un repositorio de datos persistente que permita administración y respaldos.
- Un componente de procesamiento de datos para reportería para ejecutar agregación, filtros y exportación con capacidad para exportar la información en formato definido.
- Disponibilidad en internet a través de una dirección web, dominio o subdominio definido por la Universidad Politécnica Salesiana en donde se alojará a la solución.

El prototipo actual está desarrollado sobre tres plataformas principales: Java con Springboot, Typescript/Javascript con Angular y como persistencia MongoDB como servicio cloud. *NOTA*: Estas plataformas serán revisadas en detalle más adelante en el documento como parte del proceso de mejora de este proyecto.

2.4. Restricciones de diseño e implementación

No se han definido restricciones fuertes en los requerimientos iniciales ni en la presente revisión. El diseño e implementación de la solución puede utilizar cualquier técnica y servicio disponible que se determine en el proceso de desarrollo. Sin embargo, existen algunas características que se debe considerar para la implementación del sistema:

- De preferencia, las tecnologías deben ser multiplataforma sin requerir de un sistema operativo específico para su instalación o dependencia dura de servicios de terceros para infraestructura.
- De preferencia, se debe utilizar software de tipo open source y de libre distribución. La adquisición y uso de licencias de software comercial deberá ser evaluado con los stakeholders, de ser el caso.

- El sistema debe poder desplegarse en entorno nativo de sistema operativo así como utilizando contenedores como Docker para facilitar las tareas de administración y configuración.

2.5. Suposiciones y dependencias

- La solución debe ser de fácil administración y de despliegue relativamente fácil pudiendo ser alojada de forma local o en servicios de proveedores externos en nubes comerciales como Amazon AWS, Azure, Google u otros.
- El acceso externo al sistema podría estar controlado por un servicio intermedio como un proxy Nginx que redireccione, controle y audite los accesos externos liberando al sistema de esta responsabilidad. Esto es altamente recomendable.
- Este mismo mecanismo permite el uso de certificados SSL para el despliegue del sistema en un esquema de conexión segura.
- El acceso asegurado al sistema estará detrás de las reglas de un firewall o sistema similar que controle la apertura de puertos y exposición de servicios en internet.

Actualmente, es necesario desplegar 2 aplicaciones en tecnologías distintas, correspondientes a la capa web y la capa de servicios REST. Cada aplicación tiene dependencias de plataforma base diferentes: Máquina virtual Java v11 o superior para el backend API y entorno Node.js para la aplicación web.

2.6. Tabla de implementación de requerimientos

A continuación se presenta un resumen de los casos de uso refinados, su estado de implementación en el prototipo actual y si existen mejoras propuestas por el autor y los stakeholders.

Código caso uso	Nombre	Estado Implementación	Mejoras
CU01	Iniciar evaluación de cuestionario	Implementado	SI
CU02	Responder a una pregunta	Implementado	SI
CU03	Ver resultados de la evaluación	Implementado	SI
CU04	Crear/Editar pregunta	Nuevo	
CU05	Ingreso de usuario registrado	Nuevo	

CU06	Ingreso a evaluación configurada	Parcial	SI
CU07	Crear/editar plantilla de evaluación	Nuevo	
CU08	Reportes de evaluación	Nuevo	
CU09	Administración del sistema	Nuevo	

Tabla 2. Estado de implementación de requerimientos

3.1. Revisión de Insumos Técnicos

En este documento se menciona que el código fuente está bajo control de versiones en un repositorio del sitio Github.com, sin embargo el repositorio no es accesible o ya no existe, y no se proporcionaron credenciales a este servicio que permitan corroborar su estado y el historial de control de versiones. Con respecto al despliegue y alojamiento del sistema, este se desplegó sobre la plataforma de cloud Microsoft Azure y se describe que es accesible para el público a través de una URL pública, pero al igual que el código fuente, no se proporcionaron accesos o credenciales para revisar los detalles de configuración.

Se entregaron dos archivos ZIP con los proyectos de código para dos aplicaciones independientes correspondientes a un sitio web en estilo Single Page Application y un backend de servicios REST. No se encontraron insumos adicionales de programación.

La base de datos está alojada en el servicio MongoDB Atlas, pero la única forma de acceder a la base de datos es a través de una cadena de conexión encontrada en el código fuente de la aplicación backend. Debido a esto no se pudo determinar el tipo y capacidad de este servicio, sin embargo fue posible extraer los datos existentes de la base del sistema, sacar un respaldo y desplegarlo en una máquina de desarrollo de forma local para su revisión y análisis.

Hallazgos notables

- El código entregado no tiene historial de control de versiones a pesar de que sabe que fue alojado en un repositorio de tipo Git dentro del servicio Github.

- En ambos proyectos de código se encontró un flujo de despliegue automático basado en Github actions que en este caso se encarga de compilar las aplicaciones y desplegarlas al servicio de nube Azure usando credenciales que se guardan en el repositorio Github.
- El sistema utiliza una base de datos no relacional MongoDB que permite guardar documentos con estructuras arbitrarias y jerarquías complejas en formato JSON.
- El documento original de diseño incluye un esquema JSON de la estructura del documento que se guarda por cada usuario que realiza una evaluación, pero no se detallan tipos de datos o significado de cada objeto o arreglo del documento por lo que fue necesario realizar un cierto nivel de ingeniería inversa a los datos existentes para comprender su funcionamiento y relaciones.

3.2. Tecnologías y Herramientas

De acuerdo con el documento de diseño y a una inspección inicial del código entregado se pudo determinar las tecnologías base y complementarias utilizadas en el desarrollo del sistema con el siguiente detalle:

Aplicación/Servicio	**Lenguajes y plataformas**	**Frameworks, librerías y servicios**
Frontend	Typescript Javascript adicional pnpm para dependencias	Principal: Angular v14 Adicionales: Jquery 3.5.1, Bootstrap 4.5.3, chart.js, zone.js, otros.
Backend	Java v11 Maven para dependencias	SpringBoot 2.5.5 (paquete starter para mongodb), Project Lombok, SpringFox
Base de datos	MongoDB, base NoSql	Servicio cloud MongoAtlas
Entorno de desarrollo	IDE Netbeans	No se puede determinar que complementos se utilizaron.
Versionamiento	Github	Workflow de despliegue para Microsoft Azure

Tabla 3. Tecnologías originales de desarrollo

Con esta información, se decidió utilizar las siguientes herramientas para el análisis y revisión del sistema:

- IDE IntelliJ IDEA Ultimate, 2023.1 Ultimate. Edición, depuración e Inspección de Código.
- Plugin SonarLint v7.4 para IDEs Jetbrains (IntelliJ). Análisis estático e inspección de código.

- Servidor SonarQube Community Edition v9.8 con perfiles para Java y Javascript para análisis estático, de seguridad y métricas de calidad.
- IDE para base de datos: Navicat Premium 16.
- Visual Studio Code como ambiente complementario multipropósito.
- Herramientas de desarrollo de los navegadores Firefox v109.1 y Brave v1.48 este último basado en núcleo Chromium.

Las revisiones y pruebas se llevaron a cabo en varias máquinas de desarrollo, principalmente en una Workstation con CPU Intel 8700k, 32GB RAM, 2 discos SSD NVME 1TB y sistema operativo Windows 10 Professional. Adicionalmente se hicieron pruebas en Laptops de tipo x86, una basada en Windows con procesador AMD Ryzen 9 y la otra siendo una Apple MacBook Pro (late 2013), ambas con disco SSD de más de 512GB de almacenamiento y 16GB de RAM.

La IDE de desarrollo y el entorno de base de datos son herramientas de pago que fueron adquiridas como parte de la actividad profesional del autor, pero se determinó que es posible realizar las mismas tareas de análisis con las versiones de libre distribución o con otros productos similares.

Se realizaron pruebas de compilación e instalación del software entregado y se pudo desplegar las dos aplicaciones y la base de datos en las máquinas de desarrollo. Para instalar el sistema en un ambiente de producción es necesario tener instaladas las plataformas base, en este caso Node.js y Java JDK en las versiones requeridas y luego desplegar las compilaciones de despliegue de cada aplicación. La base de datos debe existir en el servidor configurado y las estructuras de datos, colecciones de MongoDB en este caso, se crean a la primera operación de agregar datos.

Hallazgos notables

- Se encontró un archivo llamado *nbactions.xml* el cual usualmente se genera al utilizar la herramienta Netbeans para desarrollar software en Java. Adicionalmente, en el proyecto web, existe un archivo llamado ".editorconfig" el cual se usa para configurar opciones y

estilos de edición de código y funciona en la mayoría de IDEs actuales como Visual Studio Code o el mismo Netbeans.

- Con una instalación base de Node.js, el comando npm para descargar dependencias falla con advertencias sobre conflictos de versiones. Es posible forzar la instalación.
- Se encontró un archivo llamado pnpm-lock.yaml el cual se usa con el gestor de paquetes pnpm (https://pnpm.io) en proyectos para Node.js como una mejor alternativa al gestor npm que viene por defecto.
- En la estructura de archivos y directorios se observa que ambos proyectos fueron creados a partir de las plantillas base para proyectos de Angular y SpringBoot para frontend SPA y aplicación web respectivamente, con cierta personalización adicional. El proyecto web incluso conserva el archivo README.md por defecto de la herramienta de creación.
- Ambos proyectos contienen flujos de despliegue CI/CD para plataformas de nube los cuales son utilizables solo si el proyecto de aloja en repositorios del servicio GitHub.

3.3. Diseño y Arquitectura

En esta sección se detallan las observaciones al diseño, arquitectura y estructuras de datos de los componentes de la aplicación además del estado de despliegue del sistema.

En el documento original del sistema de detección de phishing, se describe el diseño de la arquitectura del sistema en tres niveles: Cliente o frontend, Servidor o backend y base de datos, haciendo referencia a las tecnologías descritas en la sección anterior respectivamente. Cabe recordar que se menciona que el segundo nivel (backend) contiene la lógica de negocio. Adicionalmente, se menciona que parte del diseño está basado en una arquitectura de microservicios.

A un nivel general, la revisión del sistema realizada para este proyecto confirma que el sistema está dividido en tres capas físicas que constan de dos aplicaciones distintas, frontend y backend que se comunican a través de servicios REST y finalmente el backend habla con la base de datos usando protocolo nativo.

Se crearon diagramas ilustrativos de la arquitectura del sistema utilizando la herramienta structurizr (https://structurizr.com) que permite crear diagramas como código en el estándar C4 (https://c4model.com). Los siguientes diagramas muestran la estructura general de aplicación de detección de phishing a nivel de contexto de sistema y macro contenedores:

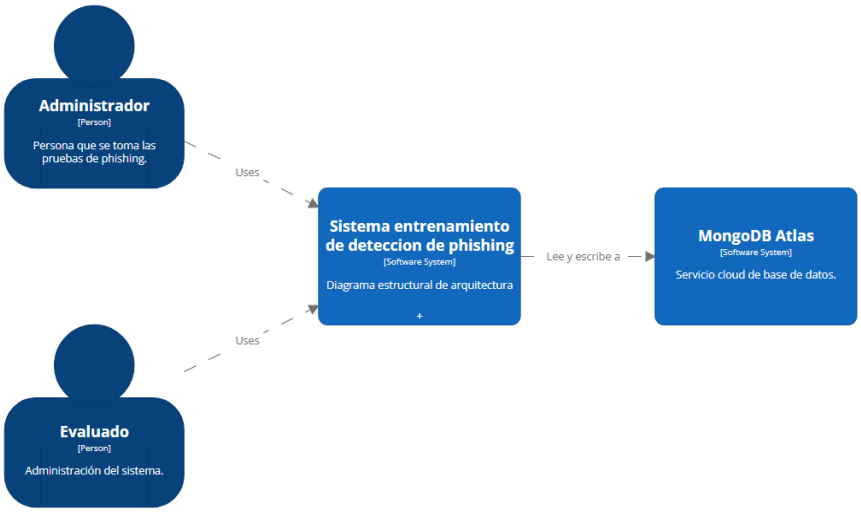

Ilustración 1. Arquitectura A Nivel Contexto (Sistema Actual)

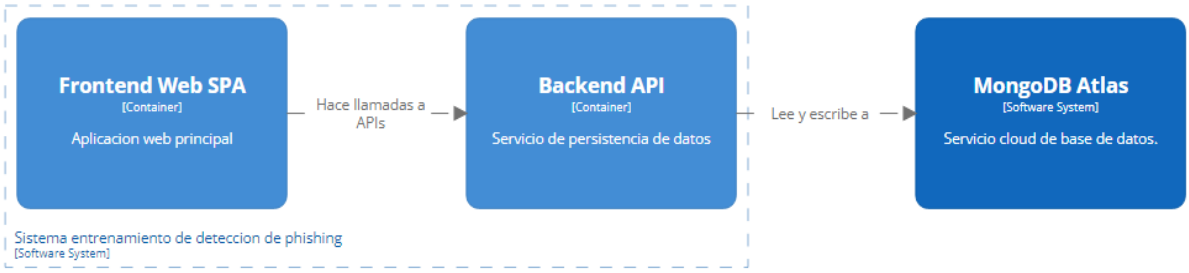

Ilustración 2. Arquitectura A Nivel Contenedores (Actual)

Teniendo en cuenta esta división en la arquitectura, se realizó una revisión más profunda de la estructura general del sistema y luego de la organización interna de cada aplicación así como las estructuras de datos utilizadas.

Arquitectura general

La separación definida entre interfaz de usuario y capa de negocios en aplicaciones distintas es una forma de arquitectura ampliamente utilizada en muchos tipos de sistemas modernos donde se permite la evolución de cada aplicación por su cuenta y se facilita el uso de tecnologías distintas que se comunican a través de protocolos y formatos estándar como HTTP y JSON.

Despliegue

Se conoce que las aplicaciones fueron desplegadas en Microsoft Azure pero no se tienen otros detalles adicionales más que la URL https://test-phishing-api.azurewebsites.net/ encontrada como endpoint para la aplicación backend. La aplicación se encontraba funcional según reporte de usuarios; sin embargo, al momento de la edición de este documento, la URL reporta que la aplicación se encuentra detenida con un código HTTP 403. En la página se recomienda que se contacte con algún administrador y muestra enlaces al portal de Azure para resolver problemas. Este comportamiento sugiere que existe algún problema con la cuenta de Azure, ya sea que se dio de baja la aplicación para reducir costos o la cuenta entera está en desuso.

Escenarios alternativos

En el código actual se observó que existen dos proyectos de software distintos para cada aplicación, cada uno con su propio versionamiento en Github y procesos de despliegue distintos.

Si este flujo genera complejidad, existe la posibilidad de unificar ambos proyectos en uno solo lo cual permitiría tener un solo repositorio de código y trabajar las características utilizando ramas distintas. Las herramientas de desarrollo modernas permiten tener código de frontend y backend en el mismo proyecto separados por directorios e integrados durante el proceso de construcción y despliegue.

Por otro lado, el uso de una aplicación SPA no está enteramente justificado, ya que no se ha estimado una gran cantidad de usuarios o la necesidad de una interfaz enteramente fluida al tratarse de un test de percepción y la presentación de resultados.

Aplicación frontend

La aplicación web que se presenta a los usuarios es de tipo Single Page Application usando el framework Angular con librerías adicionales. Se utilizó el lenguaje Typescript el cual se compila y distribuye en un bundle para ser ejecutado en un navegador de internet. Este proceso de compilación requiere del servidor Node.js instalado en la máquina de desarrollo o en un flujo de despliegue CI/CD.

El framework Angular funciona con base al concepto de componentes donde cada uno de estos puede ser una página web o partes de esta las cuales se actualizan dinámicamente con datos generalmente recibidos de un backend. Cada componente sigue por convención el modelo MVC: Modelo, Vista y Controlador con archivos distintos generados para la vista, estilos, controlador y test unitario.

Se observó que los archivos de código están organizados de acuerdo con las convenciones de Angular, mientras que el código adicional está en carpetas externas. Se utilizó la capacidad del framework para incluir dependencias declaradas en otros componentes instrumentados así como inclusión a través de importación.

Dentro de estos archivos, se encontró uno llamado constants.ts que, además de contener ciertas contantes del sistema, tiene estructuras con los detalles de las preguntas del test de phishing incluyendo el tipo de pregunta, descripción, valoración, complejidad y enlaces a imágenes relacionadas expresado como arreglos y diccionarios de código Typescript.

Realizando un seguimiento de uso de estas estructuras en el sistema con las herramientas de la IDE, se encontró que la lógica de negocio relacionada con selección de preguntas y calificación se encuentra en un componente de Angular llamado PlayViewComponent expresado como una secuencia de sentencias if y accesos directos a índices de las estructuras mencionadas anteriormente. Este componente tiene como vista una página html extensa donde se puede encontrar el contenido literal de cada pregunta, es decir el texto formateado de los mensajes de

email que el usuario verá durante el test, instrumentado con condicionales para desplegar la pregunta de acuerdo al número de ejercicio actual.

Esta revisión comprueba que efectivamente, la aplicación frontend está utilizando un archivo html y código estático como la base de datos de preguntas del sistema y que además, contiene la lógica de negocio y los algoritmos de selección de preguntas en la capa de presentación.

Existe una página de administración donde se muestran los resultados de forma básica pero no se puede ver el detalle de estos. Esta página tiene un control de acceso con nombre de usuario y contraseña, sin embargo estas credenciales están quemadas en el código del controlador de Angular y su funcionalidad es muy limitada.

Despliegue

Utilizando la configuración de despliegue se generó la versión de distribución de la aplicación. Con todos los insumos incluidos, el código JavaScript compilado ocupa 1.16MB y el punto de entrada es el archivo index.html como es usual en aplicaciones SPA de Angular.

Existen algunas advertencias del compilador pero están no se refieren al código principal del sistema sino a bibliotecas y no tienen incidencia en el proceso. Para las pruebas del sistema, se desplegó en una instancia de pruebas del servidor NGINX v1.20.2 en la máquina de desarrollo del proyecto. No se encontraron problemas con el despliegue ya que consiste básicamente en copiar los archivos generados a la carpeta raíz del servidor web o contenedor donde se alojan y configuran las rutas en el servidor web.

Se intentó correr la aplicación dentro de una carpeta en el servidor web, incluso cambiando la configuración de redirección de peticiones pero esta no funcionó correctamente ya que las rutas, como están configuradas, asumen siempre un despliegue en la raíz.

Navegación y responsividad

Para pruebas generales en un ambiente de usuario se realizaron con varios navegadores web con resultados similares. Para la presentación de la información se muestra la salida de las herramientas de desarrollo del navegador Firefox 110.0.1 con la simulación de varios dispositivos y resoluciones de pantalla.

La carga inicial de la aplicación representa una descarga de 1.58MB con todos los insumos lo cual está dentro de los promedios para sitios web actuales. El tiempo de descarga depende de la conexión del usuario por lo que está fuera de la evaluación ya que no se detectaron anomalías en los tamaños de los recursos para el navegador.

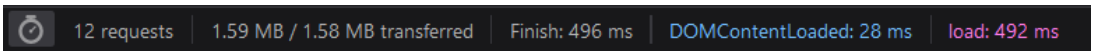

Ilustración 3. Datos De Carga De Aplicación Web

A continuación se muestran algunas de las pantallas de la aplicación en varios modos de visualización, en modo vertical (portrait) y horizontal (landscape)

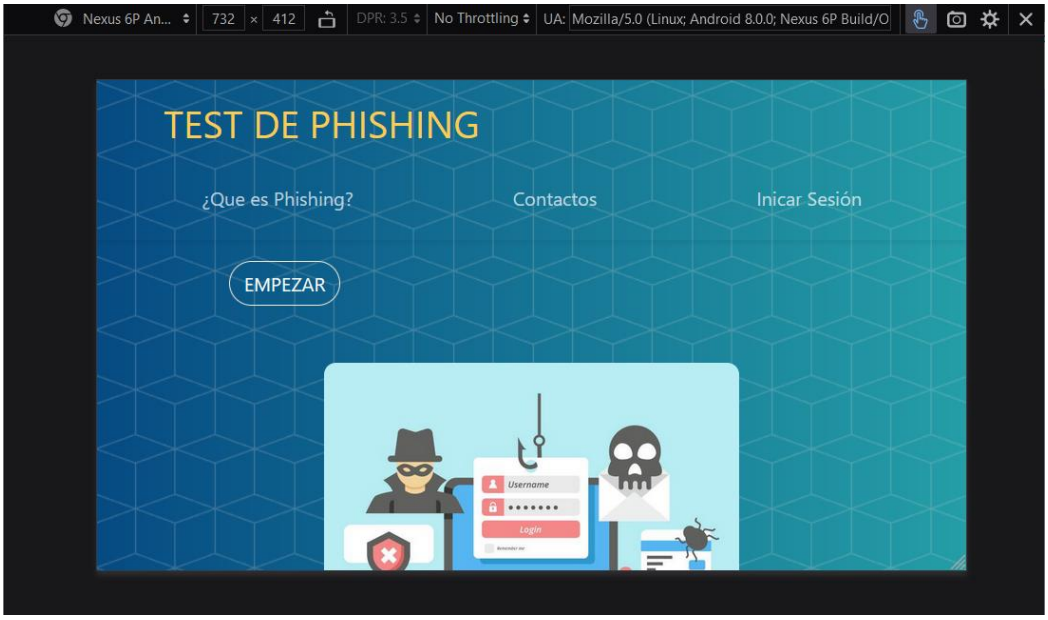

Ilustración 4. Página Inicial. Celular Android Landscape

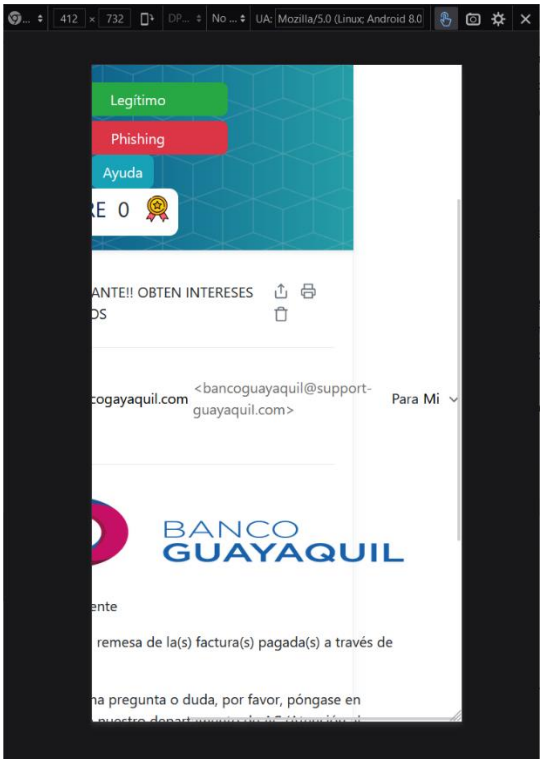

Ilustración 5. Página De Preguntas. Celular Portrait (Vertical)

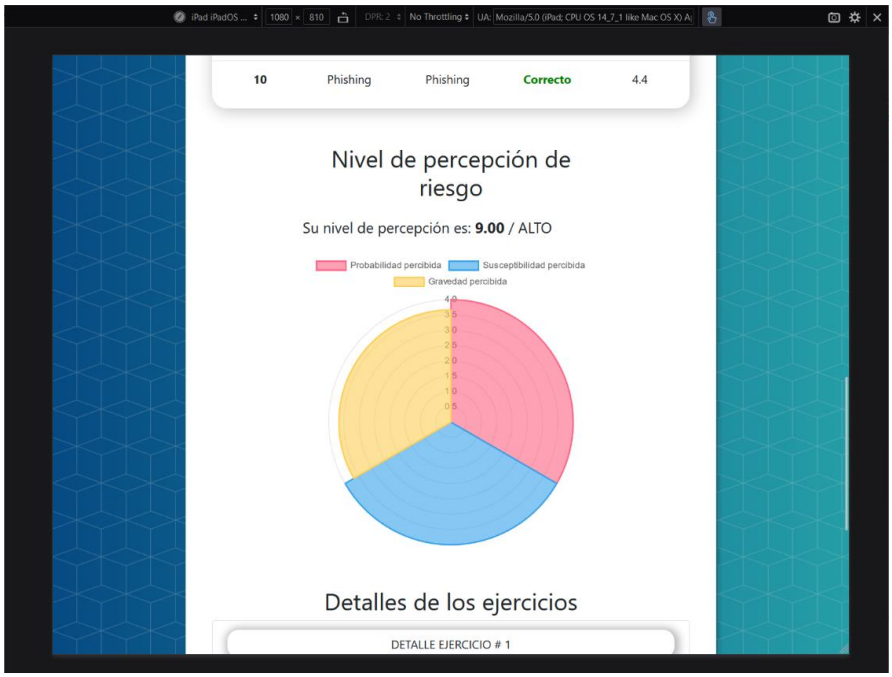

Ilustración 6. Página De Resultados. Ipad Landscape

La mayoría de las páginas se despliegan de forma correcta en varios dispositivos con la excepción de las páginas de preguntas en celulares en modo vertical donde se observa que algunas imágenes y elementos quedan fuera del cuadro de vista principal.

Adicionalmente se encontraron muchos mensajes de depuración los cuales deberían ser filtrados en un entorno de producción.

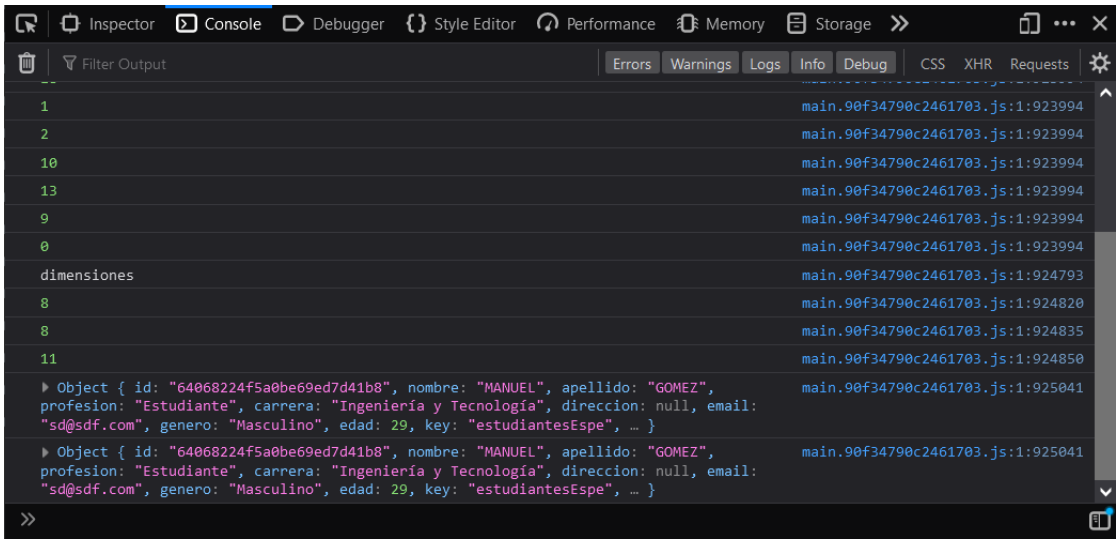

Ilustración 7. Salida De Consola Depuración Javascript

Hallazgos notables

- La URL del servicio backend está quemada como una constante y si esta cambia, se requiere una nueva compilación y redespliegue de la aplicación.
- La aplicación se apoya en las bibliotecas Bootstrap para diseño del sitio además de los recursos de una plantilla llamada “new-age” disponible en internet.
- Además de los estilos de la plantilla no se encontraron archivos de estilos para todo el sitio, solo estilos específicos a ciertos componentes lo que presenta un alto grado de duplicidad de código.
- Se encontró un uso extenso del componente localStorage del navegador para guardar varios tipos de información, desde el perfil de usuario hasta datos del cuestionario en curso, respuestas y avance del usuario evaluado.

- Muchos componentes contienen archivos de código generados por defecto por una herramienta con contenido vacío.
- Se encontró un solo test unitario que prueba el validador de un formulario. La IDE reporta error en esta clase.
- La base de datos de preguntas está incrustada en la aplicación y esta se incluye en el bundle de compilación lo cual incrementa significativamente el tamaño de la aplicación.
- La interfaz de usuario tiene cierto grado de responsividad gracias al uso de la plantilla Bootstrap. Los mejores resultados se ven en las páginas de inicio, el formulario de registro y la lista de resultados, sin embargo las páginas operativas del test de phishing no se muestran correctamente con elementos fuera el espacio de visión principal.
- Los mejores resultados se dan con dispositivos en modo landscape (horizontal).
- La aplicación se diseñó para ser servida desde la raíz del sitio web o contenedor donde se despliega sin direcciones relativas. Los vínculos y rutas son fijas con relación a la raíz del sitio de despliegue y no puede ser utilizada desde un directorio en el servidor web.
- La aplicación tiene un flujo definido de navegación pero esto no es muy aparente para el usuario al no existir enlaces entre las secciones principales del sitio, en especial cuando se ha ingresado a la prueba de phishing donde no hay forma evidente de navegar entre las preguntas lo cual puede ser una decisión de diseño.
- En dispositivos, el encabezado superior es fijo y no tiene un mecanismo de auto ocultamiento o redimensión (p.ej. menú tipo hamburguesa). El tamaño relativo de este encabezado es muy grande en comparación al resto del contenido en especial en modo horizontal.
- La página de resultados posterior a terminar el test de phishing muestra bastante información, sin embargo esta es la única ocasión en donde un usuario puede acceder a esta información y no existen enlaces adicionales para ver este contenido.
- El formulario de ingreso de datos no tiene una correcta visualización de errores. Cuando un campo requerido está incompleto no se resalta y solo se muestra un mensaje genérico que dice que el formulario no es correcto.

Aplicación backend

La aplicación backend está desarrollada en el lenguaje Java usando el framework SpringBoot que permite la creación de aplicaciones de todo tipo. El framework permite un desarrollo relativamente rápido si se utilizan ciertas convenciones definidas que automatizan la creación de clases de acceso a datos, inyección de dependencias y publicación de servicios web entre muchas otras funcionalidades. Para la gestión de dependencias y tareas de construcción se utiliza la librería Maven, estándar en la industria y la cual está incorporada en la mayoría de los entornos de desarrollo modernos.

El proyecto sigue las convenciones de un proyecto web estándar generado con la misma herramienta de SpringBoot utilizando librerías adicionales como un driver de conexión a MongoDB y SpringFox para generación de documentación automática de servicios REST en el estándar OpenAPI, conocido también como Swagger.

La estructura de directorios, sus nombres y el contenido de los archivos de clases denotan la intención de una arquitectura interna de tres niveles o capas con responsabilidades definidas de la siguiente forma:

- Interfaz externa. Exponer los datos al usuario externo, en este caso usando servicios web REST en clases llamadas controladores. En su mayoría se utilizaron objetos de transferencia (DTO) que se mapean a objetos persistentes.
- Servicios: Clases que representan la abstracción de la funcionalidad principal del sistema. Son utilizadas por la capa de interfaz y se comunican entre sí y con la capa de acceso a datos
- Acceso a datos: Abstracción sobre el repositorio de datos que permite consultas y actualizaciones de objetos persistentes. Se utilizó la autogeneración de Spring para su implementación

Este estilo arquitectónico es muy utilizado en sistemas medianos o grandes con muchos servicios que interactúan entre sí y con requerimientos de procesamiento de datos extensos ya que en los

servicios se puede construir lógica compleja con varias dependencias que solo exponen funcionalidad concreta a la capa de interfaz o a otros servicios externos.

En la revisión del código de este proyecto se encontró que existen dos servicios, los cuales actúan solo como fachada para las clases de persistencia de datos, salvo un método que contiene un cálculo el cual se usa al momento de guardar el registro de una evaluación al final de esta. No se encontró lógica adicional de cálculos, decisiones o flujos relacionados con el negocio.

Una buena mayoría del código revisado es “boilerplate” o código repetitivo necesario para que funcionen las herramientas pero que no tiene valor de negocio implícito. Las porciones de código más relevantes se encuentran en dos paquetes: model, que contiene las clases persistentes y que permite comprender la estructura de los datos almacenados y el paquete controllers que contiene el API para consumir externamente y permite ver la definición de los servicios web.

Hallazgos notables

- En el código se evitó la creación explícita de las funciones Getter y Setter en clases de datos, considerado código repetitivo. Esto se logró gracias a la biblioteca Lombok que genera esto en bytecode al momento de compilación y es transparente para el usuario.
- No se encontró ningún mecanismo de control de acceso a las APIs publicadas siendo estás todas totalmente públicas.
- Todo el código está contenido dentro del paquete base: ec.edu.espe.tphishing.testphishing.
- Es posible obtener una página de documentación del API de los servicios REST a través de la URL “/api” desde la raíz del sitio del backend.
- En las clases de configuración del framework, se encontró que la cadena de conexión a la base MongoDB está quemada como código estático.
- No se encontró el uso de mecanismos de configuración de variables.
- Existe una Excepción personalizada que se usa en un solo punto del sistema.

Base de datos

El almacén principal de datos utilizado es una base documental MongoDB. MongoDB es un servidor de base de datos que permite almacenar documentos con estructuras arbitrarias dentro de grupos llamados “colecciones”. Cada elemento almacenado tiene un identificador único y puede tener relaciones con otros aunque se estila guardar toda la información relacionada en el mismo documento en forma de una jerarquía de objetos y arreglos internos.

MongoDB trabaja principalmente con el formato JSON para representar la información y las consultas pueden realizarse a través de comandos o con las funciones específicas del conector de cada lenguaje y plataforma soportada.

Este servidor está disponible para la mayoría de los sistemas operativos actuales así como en servicios SaaS de distintos proveedores de nube. En el caso del sistema analizado para este proyecto, se utilizó el servicio MongoDB Atlas que provee instancias de MongoDB manejadas disponibles en internet. Anteriormente se describió que solo se tiene acceso a la instancia del proyecto a través de una cadena de conexión que permite ver las bases de datos creadas pero no la configuración y capacidad de esta instancia por lo que este tema se deja fuera del análisis.

Por parte de los datos, se observó que la colección principal llamada “perfiles” contiene toda la información útil del sistema en donde cada documento representa la evaluación de un test por un usuario y contiene el orden de las preguntas, las respuestas con los tiempos y los totales. Esta estructura es la única que posee alguna descripción en la documentación original, si bien es posible ver y editar el documento JSON a través de una IDE y obtener el mismo resultado a través de observación.

Hallazgos notables

- La base de datos actual contiene 3 colecciones de las cuales solo 2 se referencian en el backend.
- Las colecciones identificadas son:

  - codigos-clase: contiene un campo operativo con el nombre de lo que se asume es el tipo de usuario.
  - perfiles: contiene en un solo documento toda la información personal del usuario que toma el test, las preguntas, los tiempos de respuesta, las respuestas y los totales acumulados. Esta es la única estructura con documentación mínima.
- No se encontraron índices sobre las colecciones.
- MongoDB crea las estructuras si estas no existen al momento del primer ingreso.

3.4. Análisis Estático

Cada una de las dos aplicaciones utilizan tecnologías diferentes y es necesario analizarlas dentro del contexto de cada una de ellas, sin embargo existen características comunes como que, en este caso, ambos lenguajes utilizan clases, orientación a objetos y estructuras imperativas similares con lo que varias métricas obtenidas de las herramientas pueden ser analizadas con un criterio de calidad de software general.

El análisis estático del código se realizó principalmente con la herramienta SonarQube, con perfiles de calidad de software estándar para proyectos Java y Javascript/Typescript. Adicionalmente se utilizó el reporte de inspección de código de la interfaz Intellij IDEA y del plugin SonarLint en la misma herramienta.

Cada aplicación fue analizada de forma independiente y a continuación se presentan los resultados empezando con una visión general del código para luego ver una lista de las incidencias encontradas y determinar cuales tienen más impacto en la calidad y/o funcionalidad de los componentes afectados.

Métricas utilizadas

A continuación se expone una breve descripción sobre las métricas utilizadas en las mediciones para facilitar su comprensión, de acuerdo con la documentación de SonarQube encontrada en la siguiente url: https://docs.sonarqube.org/latest/user-guide/metric-definitions.

- **Líneas de código y Líneas totales**: Número de líneas de texto que contienen código fuente y líneas de texto en general.
- **Clases y funciones**: Número de clases y número de funciones o métodos
- **Complejidad ciclomática**: Medida cuantitativa de complejidad usada para calcular el número de posibles caminos que puede seguir el código. Siempre que el flujo de control se divide, el contador de complejidad incrementa. Existen variaciones por cada lenguaje.
- **Complejidad cognitiva**: Qué tan complejo de entender es el flujo de control de una función. A grandes rasgos se calcula en función del número de condicionales y del número de condiciones y bucles anidados.
- **Archivos**: Número de archivos analizados
- **Relación de Deuda técnica**: Relación entre el costo de desarrollar software vs el costo de arreglarlo. La fórmula es: Costo de remediación / (Costo de desarrollar 1 línea de código * # líneas de código). El costo de desarrollo de una línea de código es de 0.06 días, según la documentación de SonarQube.
- **Deuda técnica en tiempo (t)**: Medida de esfuerzo para arreglar todos los code smells encontrados. Se asume un día de 8 horas laborables.

Para la medición de incidencias se consideraron las siguientes métricas:

- **Errores**: Fallas de programación o posibles condiciones de error en el código.
- **Code smells**: Total de incidencias de código sospechoso
- **Vulnerabilidades**: Referencias a código o componentes con vulnerabilidades conocidas.
- **Líneas duplicadas**: Número de líneas de código con repetición
- **Líneas duplicadas %**: líneas duplicadas / total de líneas de código
- **Esfuerzo de remediación**: El esfuerzo de arreglar las incidencias, usando el criterio de deuda técnica

Aplicación frontend

Estadísticas del código			
Métrica	**Medición**	**Métrica**	**Medición**
Líneas de código	7007	Líneas totales	7676
Clases	25	Funciones	130
Complejidad ciclomática	185	Complejidad cognitiva	67
Relación de Deuda técnica	1.4%	Deuda técnica (t)	6d 2h
Archivos	68		

Tabla 4. Resumen métricas código frontend

Incidencias			
Métrica	**Medición**	**Métrica**	**Medición**
Errores	78	Líneas duplicadas	1616
Code smells	411	Líneas duplicadas %	21%
Vulnerabilidades	0	Esfuerzo de remediación	4h 43 min

Tabla 5. Incidencias código frontend

Aplicación backend

Estadísticas del código			
Métrica	**Medición**	**Métrica**	**Medición**
Líneas de código	588	Líneas totales	906
Clases	19	Funciones	35
Complejidad ciclomática	40	Complejidad cognitiva	19
Relación de Deuda técnica	0.5%	Deuda técnica (t)	1h 26min
Archivos	20		

Tabla 6. Resumen métricas código backend

Incidencias			
Métrica	**Medición**	**Métrica**	**Medición**
Errores	1	Líneas duplicadas	0
Code smells	16	Líneas duplicadas %	0
Vulnerabilidades	2	Esfuerzo de remediación	10min

Tabla 7. Incidencias código backend

Observaciones

La aplicación frontend contiene la mayor cantidad de código, dividido en código Typescript de Angular, plantillas html y estilos CSS. Mucho del código Angular es repetitivo y puede ser refactorizado o eliminado sin problema.

Este mismo sistema posee la mayor complejidad ciclomática y cognitiva ya que alberga la lógica principal de negocio como se advirtió en la revisión manual. El problema detectado con la base de datos de preguntas se hace evidente al observar el alto número de líneas repetidas que se concentran en el archivo HTML que contiene la data de las preguntas del test. Este mismo archivo posee la mayor cantidad de errores y code smells ya que grandes porciones de HTML con defectos también están repetidas en la definición de la vista de cada pregunta.

Este alto nivel de repetición distorsiona los resultados de otros componentes también afectados por code smells, sin embargo, al filtrar estos datos se observa que la resolución de estos problemas en código Typescript es relativamente más fácil y no se observan vulnerabilidades de seguridad.

Con respecto a la aplicación backend se observa un bajo nivel de incidencias relativas a todo el sistema. Se reporta un error el cual, tras inspección manual, se refiere más a una comprobación adicional de consistencia y las vulnerabilidades reportadas son recomendaciones para cerciorarse que las APIs tengan el nivel de acceso adecuado por políticas CORS. Como se observó en la revisión manual, la aplicación backend no tiene casi nada de lógica de negocio por lo que sus medidas de complejidad son bajas, así como la deuda técnica y esfuerzo de remediación según la herramienta. Sin embargo, al hacer una relación entre el número de líneas de código (considerando también número de clases y funciones) contra las medidas de complejidad, se puede observar que esta aplicación contiene básicamente código ocioso y repetitivo principalmente debido a la plataforma utilizada, incluso con los esfuerzos de reducirlo por parte de los desarrolladores originales.

3.5. Conclusiones del Análisis

Teniendo en cuenta el alcance del sistema actual, el uso de dos aplicaciones distintas (con repositorios y despliegues distintos) introduce esfuerzo adicional para el cual no se encontró justificación más allá de ser una arquitectura moderna que prueba efectivamente las capacidades de las herramientas.

El principal defecto encontrado en el sistema es sin duda la forma como se almacenan y utilizan las preguntas del test de phishing en la interfaz de usuario usando código Typescript y HTML de interfaz de usuario como base datos existiendo claramente alternativas a este diseño.

El segundo problema encontrado es la organización de la lógica del sistema y la división de responsabilidades. Todos los algoritmos de decisión relacionados con la evaluación de las preguntas del test se encuentran en la capa de frontend, lo cual está directamente relacionado con el punto anterior al tener los datos operativos en un solo lugar. Incluso si la data de preguntas fuera enviada desde el backend, el hecho de tener lógica de negocio mezclada con lógica de presentación y datos se considera una pobre decisión de diseño conocido como “código spaghetti” (Fowler, 2018).

La falta de pruebas unitarias puede considerarse como una falla en utilizar buenas prácticas más que un defecto en código debido a que este tipo de pruebas ayuda para la verificación de funcionalidad y consistencia del sistema, sin ser un requerimiento determinante para el éxito de una aplicación. Sin embargo, la experiencia colectiva en desarrollo de software ha demostrado que estas pruebas ayudan a mejorar la calidad del software a la larga (Fowler, 2018) y muchas herramientas de análisis actuales mostrarán advertencias si no se encuentran pruebas de algún tipo en el código por lo que es recomendable crearlas.

La falta de estas pruebas puede ser mitigada creando código para probar las secciones críticas del sistema, sin embargo, esto se dificulta debido a algunas de las decisiones de diseño mencionadas en los párrafos anteriores.

En la revisión se detectaron otros problemas menores como código inútil que puede ser eliminado de forma automática por las herramientas o manualmente por el equipo ya que, como se ve en las estadísticas del análisis estático, la base de código no es grande en este caso.

3.6. Propuesta de Rediseño

Recomendaciones de refactorización

Una vez identificados los problemas que deben ser resueltos es necesario determinar cuáles podrían ser las soluciones más adecuadas para cada uno de estos. Con base en las recomendaciones de las herramientas, la literatura disponible y la experiencia profesional del autor en buenas prácticas de desarrollo, se encuentra que las siguientes refactorizaciones y cambios son requeridos:

Aspectos generales

- Volver a poner cada proyecto bajo control de versiones, incluso, una mejor opción es poner ambos proyectos bajo el mismo repositorio para reducir carga operativa. El mantener un control de versiones del código es fundamental en entornos de desarrollo modernos y este debe estar presente incluso en proyecto de desarrollo de corto alcance.
- Remover código inútil. Se encontró muchas clases, métodos y archivos vacíos o con contenido autogenerado que no tiene funcionalidad real alguna y sube el tamaño de la base de código sin agregar ningún valor.
- Eliminar repetición de código. Se encontró en varias partes del sistema, en especial en la aplicación frontend.
- Actualizar las librerías a las versiones más actuales y estables, con el fin de minimizar riesgos de seguridad y reducir la deuda técnica que supone utilizar software potencialmente obsoleto.
- Parametrización de variables de sistema. En ambas aplicaciones es necesario extraer las variables de operación como cadenas de conexión, credenciales y otras a mecanismos de configuración externa como variables de entorno, archivos de tipo .env o registros en base de datos.
- Implementar pruebas de funcionalidad crítica automatizadas. Las aplicaciones actuales no contienen pruebas válidas que permitan verificar ningún cumplimiento de

funcionalidad de negocio. Con base en los requerimientos y los criterios del autor, se considera que es necesario implementar pruebas unitarias de acuerdo con el contexto del componente probado y definir un nivel de cobertura adecuado para el proyecto. Adicionalmente, se recomienda crear pruebas de integración y de performance (si se requiere) las cuales podrían integrarse en un flujo CI/CD posteriormente.

- Introducir comentarios relevantes en el código opcionalmente. Durante la revisión manual de varios componentes del sistema se observó que si bien la mayoría de los nombres de variables y estructuras denotan su intención y son explícitas, no existen comentarios sobre el funcionamiento o propósito de los algoritmos de cálculos y flujos de decisiones.

Aplicación frontend

- Extraer la lógica de negocio de esta aplicación. La revisión manual del código demostró que la lógica principal de negocio, es decir los cálculos para presentar preguntas a los usuarios, calcular resultados de los tests y dirigir el flujo del examen se encuentra completamente en la capa de interfaz de usuario dentro del código de controladores Angular lo cual introduce responsabilidades adicionales al código de UI, aumenta la complejidad de este sistema y va en contra de muchos principios de diseño y buenas prácticas de desarrollo ampliamente conocidas. Esta lógica debe ser reubicada y reprogramada en el backend de la aplicación lo cual permitirá que el frontend se especialice en mostrar y formatear la información para el usuario de mejor forma.
- Extraer la data de preguntas a una base de datos. Se detectó un severo problema de mantenibilidad y diseño en el sistema al observar que se utilizó al frontend como base de datos para las preguntas del test. Toda información dinámica de gran tamaño debe almacenarse en un storage adecuado para la tarea y no mezclarse con el código operativo, menos con el de interfaz de usuario.

  NOTA: Se estimó que esta corrección tiene un esfuerzo un poco mayor a otras debido a la necesidad de crear scripts especiales que permitan extraer el contenido HTML de cada

pregunta, empatarlos con las constantes y guardarlos en estructuras adecuadas en la base de datos las cuales también deberán ser creadas. Adicionalmente, esta refactorización involucra adecuar al backend con funcionalidad para acceder a la nueva información a petición y crear alguna forma de administración de estos datos.

- Minimizar el uso de localStorage. Este mecanismo es muy útil en aplicaciones web para guardar información temporal no sensible en el navegador web de forma global, sin embargo se detectó un problema potencial de filtración de información sensible en esta funcionalidad que incluye datos de negocio como los detalles de los cuestionarios e información de registro de usuarios los cuales no deberían guardarse en el navegador del usuario. Existen otras opciones para guardar datos transitorios, principalmente utilizando llamadas al backend del sistema.
- Eliminar estilos repetidos y combinarlos en una hoja general para el sitio. Se encontró que muchas directivas CSS se encuentran repetidas en los archivos específicos de componentes individuales y estos podrían extraerse y ponerse en un solo archivo común lo cual mejora la mantenibilidad y consistencia del aspecto gráfico del sitio web.

Aplicación backend

- Implementar la lógica de negocio principal en el backend. De acuerdo al diseño arquitectónico propuesto originalmente, la lógica de negocio incluyendo cálculos, controles y flujos debe estar en el componente backend y no en la interfaz de usuario como ya se mencionó anteriormente.
- Reprogramar los algoritmos principales. En la inspección de los algoritmos de negocio se encontró mucho código repetido, sentencias if anidadas y otros code smells que si bien podrían ser resueltos con refactorización, se encuentran en la capa de frontend y deben ser removidos de ahí. Con base en el documento de diseño original y con la retroalimentación de los stakeholders es posible recrear esta lógica en el backend donde corresponde.

- Implementar funcionalidad adicional como APIs. El paso de la lógica de negocio a esta capa supone que se creen más endpoints para la conexión con el frontend y también adaptar este a este nuevo esquema de funcionamiento.
- Revisar la recomendación de seguridad de CORS. Si se determina que la seguridad debe ser más alta para el acceso a los APIs del backend, es necesario que la configuración de Cross-Origin Resource Sharing que permite el acceso desde otros dominios esté bien definida o se implemente seguridad adicional si se va a dar acceso externo.
- Implementar seguridad de acceso externo a APIs. Al momento, la aplicación no tiene ningún mecanismo de control de acceso y cualquier usuario puede llamar a la funcionalidad y obtener o cambiar información de forma indiscriminada. Existen varias opciones como uso de tokens, JWT, OAuth o mecanismos personalizados complementarios que se pueden utilizar como limitadores de consultas y restricciones por IP.

Base de datos

- Crear una definición más detallada de las estructuras de datos persistentes. Incluso si se decide seguir utilizando MongoDB, se necesita un modelo con más claridad que denote el propósito los documentos JSON persistentes.
- Revisar la dependencia con el servicio MongoDB Atlas. Incluso si se quiere seguir utilizando este servicio, es necesario que los clientes finales posean los accesos correspondientes a la cuenta de este servicio o a cualquier otro servicio SaaS que se quiera utilizar para garantizar el acceso a los datos y la continuidad del servicio.
- Asegurar la disponibilidad de respaldos de la información. Incluso en sistemas empresariales de poco alcance, se requiere que la información sea respaldada periódicamente y se cuente con un proceso para obtener estos archivos lo que actualmente no existe.

Estimación de esfuerzo

De acuerdo con las recomendaciones propuestas para refactorización se realizó una estimación del esfuerzo necesario para llevar a cabo los cambios. Estas estimaciones son hasta cierto punto subjetivas, tomando en cuenta la experiencia profesional del autor y en menor medida, a las métricas del análisis estático de SonarQube, en especial deuda técnica y remediación las cuales están más orientadas a la corrección automatizada de los defectos sin considerar el contexto del componente o las repercusiones y riesgos en el proceso de desarrollo.

Cabe mencionar que muchas de las tareas están relacionadas y son dependientes, por ejemplo la extracción de la lógica del frontend va a la par con la implementación de esta lógica en el backend y supone cambios más profundos en los sistemas como revisar dependencias y crear nuevas conexiones entre componentes y/o aplicaciones. Para esta estimación no se consideran características nuevas de los requerimientos refinados más allá de la necesidad de administrar la data extraída del frontend.

# Tarea	Tarea	Componente	# Horas estimadas de desarrollo	% relativo en esfuerzo total
1	Control de versiones	Sistema	2	1%
2	Remover código inútil	Sistema	4	2%
3	Eliminar repetición de código	Sistema	5	3%
4	Actualizar las librerías	Sistema	3	2%
5	Parametrización de variables	Sistema	2	1%
6	Creación de pruebas	Sistema	10	5%
7	Introducir comentarios	Sistema	2	1%
8	Extraer lógica de negocio	Frontend	25	13%
9	Extraer la data de preguntas a una base de datos	Frontend	30	16%
10	Minimizar el uso de localStorage	Frontend	10	5%
11	Eliminar estilos repetidos	Frontend	6	3%
12	Implementar la lógica de negocio	Backend	15	8%
13	Reprogramar los algoritmos principales	Backend	15	8%
14	Revisar seguridad CORS	Backend	7	4%
15	Implementar APIs	Backend	10	5%
16	Implementar seguridad APIs	Backend	10	5%
17	Revisar modelo de datos	Base de datos	10	5%
18	Verificación y pruebas de cambios	Sistema	20	11%
		TOTAL	186	100%

Tabla 8. Estimación de esfuerzo de refactorización

En resumen, la estimación de los cambios contando pruebas de validación tomará un **total de 186 horas** lo que se resume en aproximadamente 23 días de jornada laboral o un mes de trabajo combinado. Los valores más altos se relacionan con extracción de datos de preguntas y lógica de negocio con 16% y 13% respectivamente, aunque si se suman las tareas relacionadas de cambio de lugar y reprogramación de la lógica, estos representan el mayor esfuerzo con un total de 37% combinado.

Propuesta de reconstrucción

Luego de revisar los resultados de los análisis del código, las observaciones de la revisión manual y teniendo en cuenta el estado de implementación de los requerimientos y las estimaciones de esfuerzo por refactorización, se considera como buena alternativa realizar una reconstrucción del sistema con nueva tecnología y arquitectura revisada.

Se estimó que es esfuerzo para construir una aplicación de este tipo es de alrededor de 20 a 25 días laborables, lo cual es muy similar al esfuerzo de refactorización estimado en 23 días. Esta estimación se basa principalmente en la experiencia del autor en la construcción de sistemas similares a lo largo de su carrera profesional.

Una decisión de este tipo no se toma a la ligera, sin embargo, para el caso particular del prototipo del sistema de detección de phishing se consideró la mejor opción por varios factores detallados a continuación:

- La funcionalidad requerida ya se encuentra claramente definida y refinada incluyendo mejoras propuestas. No es necesario pasar por una fase de análisis de requerimientos adicional.
- El prototipo actual tiene un alcance definido y no muy amplio lo cual sirve como un estándar mínimo para un nuevo desarrollo.
- De los requerimientos refinados, la mayoría son nuevos y muchos de los que ya están implementados tienen que mejorarse con base a las ideas expresadas en el documento, sin embargo, los esfuerzos de refactorización tendrían que hacerse primero antes que las

mejoras. En una reconstrucción, estas funcionalidades de desarrollan desde un principio sin pasar por cambios previos.

- Las tecnologías utilizadas en el prototipo se prestan mejor para proyectos más grandes, debido a su complejidad, requerimientos de recursos y convenciones de la industria.
- Por el mismo motivo anterior, se podría decir que estas herramientas están subutilizadas en una aplicación de este tamaño.
- Según las estimaciones realizadas, la refactorización y corrección de los problemas principales tomaría un tiempo muy similar a la reconstrucción del sistema utilizando otras plataformas más accesibles y rápidas así como un diseño y arquitectura más acordes con las necesidades planteadas.
- En especial, la extracción de las preguntas de la aplicación frontend a una base de datos es una tarea que debe realizarse de cualquier forma por lo que es menor esfuerzo guardar esta data en una nueva estructura antes que migrarla posteriormente.
- Los estilos gráficos y de presentación del sistema así como otros insumos de interfaz de usuario se podrían reutilizar e incluso mejorar en una reconstrucción.
- El modelo de datos actual no es muy amigable para la extracción y análisis de información por herramientas de reportería incluso desde el mismo sistema por lo que es necesario revisarlo de todas formas.
- El control de versiones del proyecto tendrá un historial desde el principio. En el código actual, no existe este historial y si se incorpora, solo habría registros a partir de la refactorización.
- Todas las observaciones aplicables a una refactorización estarán presentes desde un principio, evitando cometer los errores encontrados.
- El sistema actual es un prototipo funcional, una prueba de concepto que demuestra ser factible y una nueva versión mejorada afirma la validez y el propósito de la herramienta.

3.7. Planificación del Desarrollo

Uno de los resultados del análisis del presente proyecto es un detalle de tareas de refactorización necesarias para mejorar el sistema las cuales, en la práctica, se traducen en tareas de desarrollo de software tradicional como generación de código, actualización de base de datos, diseño web, pruebas, etc. Estas tareas tienen una prioridad, una estimación de esfuerzo y un alcance definido por lo que pueden ser incluidas sin problema dentro de la planificación de proyectos de desarrollo de software.

Con esto se sugiere que para metodologías ágiles como Scrum donde las tareas de cada iteración se planifican antes de su ejecución, es posible destinar tiempo y recursos necesarios para refactorización sin que estos trabajos se vean como casos especiales o fuera de lo común. Esto puede no ser el caso para proyectos donde se utilizan metodologías tradicionales como RUP que requieren de toda la planificación antes del desarrollo, pero ese análisis está fuera del alcance de este documento.

En el caso particular de este proyecto, se tomó la decisión de reconstruir el sistema tomando como base los resultados del análisis y el esfuerzo estimado en mitigación vs reconstrucción por lo que la definición y planificación de las tareas de desarrollo se realizó con base en las funcionales e historias de usuario definidas en los requerimientos y no en las tareas de refactorización detalladas. Para este fin, el primer paso es definir un marco de trabajo para el proceso de desarrollo.

Metodología de desarrollo

Para el proceso de desarrollo se utilizó un proceso ágil, iterativo e incremental utilizando como guía la metodología Scrum con adiciones propuestas por (Ashraf & Shabib, 2017) y otros autores. En estas propuestas se sugiere simplificar aspectos como documentar teniendo en cuenta el contexto y la importancia del tema así como incorporar activamente tareas de mantenimiento dentro de los sprints, lo cual se relaciona con los aspectos de refactorización descritos

anteriormente. A continuación se presenta un diagrama general de la metodología de desarrollo utilizada:

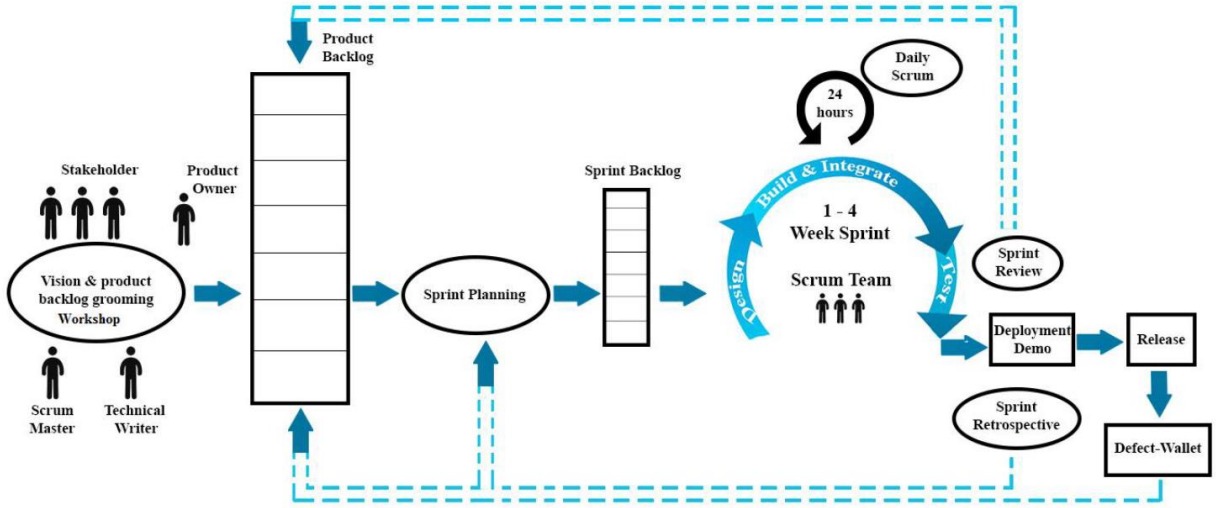

Ilustración 8. Proceso Iscrum (Ashraf & Shabib, 2017)

Se puede observar que el proceso propuesto contiene los elementos clásicos de Scrum adicionando elementos de mantenimiento como una "bolsa de defectos" ("Defect-wallet" en el diagrama) la cual sirve para mantener un registro de anomalías detectadas e incluir su corrección activamente en la planificación con la misma importancia de nuevas características.

Dentro del marco de este proyecto, se considera que esta adición de la bolsa de defectos es una mejora importante al proceso tradicional de Scrum ya que se le da mayor importancia al mantenimiento del software lo cual a la larga puede disminuir fallas así como costos de refactorización y mitigación no previstos.

Backlog del producto

El backlog del producto es la lista de características a implementarse con una prioridad definida que se deriva de los requerimientos. Para el caso del sistema de detección de phishing se definieron las características y su prioridad utilizando los procesos generales y los aspectos no funcionales detallados en la sección de requerimientos de este documento. Al ser un nuevo

sistema, se incluye la construcción y definición de elementos de la arquitectura base, lo cual es efectivamente un rediseño de la arquitectura del prototipo inicial.

En Scrum es común utilizar sprints de una semana de duración y este mismo parámetro se utilizó en la planificación del desarrollo para un total de 5 sprints, tiempo que está dentro de la estimación de la propuesta de reconstrucción.

En la siguiente tabla se detalla la planificación combinando el backlog del producto y la organización de los sprints:

# Tarea	Funcionalidad/Subsistema	Prioridad inicial	Estimación Sprints
1	Arquitectura base	Alta	1
2	Estructuras de datos	Alta	1
3	Infraestructura inicial	Media	1, 2
4	Administración de usuarios y perfiles	Media	2
5	Administración de catálogos	Media	2
6	Administración de tests y preguntas	Alta	3
8	Ingreso de usuario registrado	Alta	3, 4
9	Módulo de toma de test	Alta	4, 5
11	Reporte de resultados de exámenes	Media	5
12	Integración y despliegue	Alta	5

Tabla 9. Backlog inicial con planificación

En el cuadro se incluyeron la estimación en cuantos y cuáles serán los sprints de cada tarea del backlog. Seguidamente, se planteó un cronograma con estas fases y su duración estimada dentro de una plantilla sencilla proporcionada por el sitio https://www.onlinegantt.com/, la cual permite la creación de diagramas de Gantt de forma gratuita. En el diagrama se incluyen fines de semana principalmente en la extensión de las tareas, pero esto no implica que se haya realizado trabajo durante estos días.

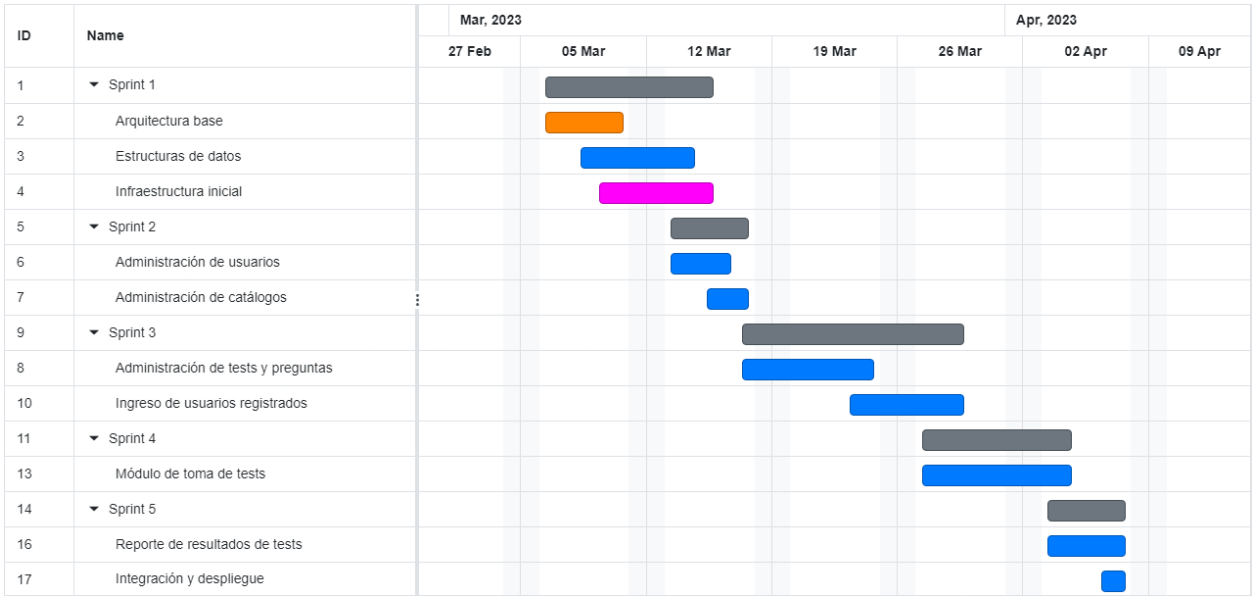

Ilustración 9. Cronograma Actividades De Desarrollo

4. Ejecución del Desarrollo

Stack tecnológico

Para el rediseño de la plataforma, se analizaron varias opciones de plataformas tecnológicas incluyendo las utilizadas actualmente y se realizaron consultas con los stakeholders del proyecto para determinar cuál opción es más adecuada considerando el tipo de proyecto, las características requeridas y el tiempo de desarrollo estimado.

Se recomendó no utilizar plataformas poco conocidas o experimentales debido a posibles temas de mantenimiento futuro y a poca compatibilidad con algunos insumos existentes en el caso de la interfaz de usuario. Para esta última, se decidió mantener ciertas bibliotecas base que son ampliamente utilizadas lo cual aporta el valor de familiaridad y disponibilidad de recursos para desarrollo como Bootstrap y Chart.js.

En el caso del backend, se decidió utilizar la versión más reciente del framework .NET de Microsoft debido a las prestaciones que posee, muy buen rendimiento, un lenguaje de programación avanzado como C# y como una gran cantidad de recursos de información, entrenamiento y consulta disponibles. Adicionalmente, se tomó en cuenta la experiencia del autor en el desarrollo de grandes soluciones empresariales en el pasado.

La nueva versión del sistema para entrenamiento de detección de phishing se construyó sobre las siguientes tecnologías.

- Plataforma base backend y web: .NET Framework 7.0
    - ASP.NET MVC, WebApi, Entity Framework
    - Gestor de paquetes Nuget y librerías adicionales
- Framework Web UI: Vue.js 3 con Javascript
    - Adicionales jQuery 3, Bootstrap 5, ChartJS entre otras
    - Gestor de paquetes npm
- Base de datos principal: PostgreSQL v14

- Sistema de control de versiones: Git
- Pruebas de despliegue: Servidor Nginx 1.20, Internet Information Server 10, Docker desktop.

En cuanto a herramientas para el desarrollo se utilizaron los siguientes programas y servicios:

- Entornos Integrados de desarrollo:
    - Visual Studio 2022 Community Edition,
    - Visual Studio Code 1.76.2 (soporte)
- Gestión de base de datos: Navicat Premium 16, pgAdmin 4

Para la gestión del proceso Scrum se utilizó la herramienta colaborativa en línea Trello de Atlassian (https://www.trello.com) con la cual se creó un tablero de gestión de backlogs y sprints del proyecto. El desarrollo se realizó en un ambiente multiplataforma, usando máquinas Windows y MacOS, en donde se encontró que la versión de Visual Studio para Mac presenta una mejor experiencia de usuario comparada con la versión para Windows, de acuerdo con la opinión del autor.

Convenciones de edición

Para el código C# existe un cambio en la convención tradicional usada para la indentación de corchetes la cual define que van en la siguiente línea de la instrucción anterior. Para este proyecto se utilizó el estilo Kernighan & Ritchie (K&R) en donde los corchetes, paréntesis y otros caracteres de puntuación van en la misma línea, similar al estilo usado en el lenguaje Java. El código JavaScript lleva el mismo estilo de esta convención por defecto.

Adicionalmente, se utilizó el carácter TAB en lugar de espacios para la indentación general en todos los lenguajes utilizados.

Si bien estas convenciones son simplemente preferencias del autor, se deberían tomar en cuenta en el caso de que, al utilizar herramientas de análisis estático de código, se reporten estos cambios como errores de estilo.

La configuración de la edición de código se exportó a un archivo de tipo ".editorconfig" el cual es compatible con la mayoría de entornos de desarrollo modernos y se incluye también en el repositorio ya que contiene las convenciones mencionadas anteriormente.

Herramientas adicionales

Adicionalmente, fue necesario el uso de herramientas adicionales para la migración y procesamiento de los datos del prototipo, así como otras pruebas. Se crearon programas para tareas como Extracción, Transformación y Carga de datos (ETL), lectura de JSON y procesamiento de texto entre otras. Concretamente se utilizó:

- Creación de scripts: PHP 8.2
- Gestión de paquetes con composer
- IDE: PhpStorm 2022.3.3.

Los programas resultantes también fueron incluidos en un repositorio de código adicional complementario al proyecto, pero no son necesarios para el despliegue o funcionamiento del sistema finalizado.

4.1. Backlog de mantenimiento

Los elementos del backlog del producto y las tareas planificadas se organizaron como tarjetas en listas dentro de un tablero creado en Trello. Se utilizó una plantilla inicial de Scrum proporcionada por la herramienta y se adaptaron otras estructuras de acuerdo con la metodología propuesta IScrum (Ashraf & Shabib, 2017), la cual propone ciertas simplificaciones a la parte administrativa del proceso de desarrollo pero adiciona una sección específica para mantenimiento continuo lo cual es ideal para incluir tareas de reparación y refactorización.

En el tablero de Scrum se creó una lista especial llamada "Defect-Wallet" que contiene la lista de las tareas de refactorización determinadas en la sección 3.6 las cuales sirven como un backlog adicional de mantenimiento que complementa al backlog de producto y al del sprint en curso.

De acuerdo con la prioridad dada, estas tareas pueden ser incluidas en la ejecución del sprint actual o, si no se encuentra resolución o cambia el enfoque del problema, pueden ser devueltas al backlog de defectos para una replanificación. De esta forma, es posible hacer seguimiento específico a ítems determinados como tareas de refactorización y mantenimiento en un proceso Scrum.

En el caso del presente proyecto, la inclusión de estas tareas sirvió para determinar en cual sprint (o sprints) del desarrollo se llegó a la solución de cada problema listado en el defect-wallet. Con este enfoque, en las siguientes secciones se presenta un detalle del trabajo realizado en cada sprint.

Cabe resaltar que al ser un proceso de desarrollo iterativo, las primeras versiones obtenidas de componentes y módulos fueron a su vez refactorizadas y mejoradas para el producto final y estos pasos se incluyeron también en el backlog de mantenimiento.

4.2. Detalle del Proceso de Desarrollo

Formato de presentación

Para presentar el desarrollo de cada sprint se creó un formato simple de documentación, similar en esencia a una narrativa contada en retrospectiva que contiene una referencia a los ítems del backlog involucrados, un detalle de las tareas realizadas, inconvenientes encontrados y detalles técnicos complementarios.

Al final del detalle de sprint, se incluye una referencia a los defectos resueltos que salen del backlog de mantenimiento, cuál fue su resolución y notas adicionales sobre el tema.

4.2.1. Sprint 1

Backlog: 1. Arquitectura base, 2. Estructuras de datos, 3. Infraestructura inicial, 6. Creación de pruebas.

Detalle y actividades:

Al inicio de esta etapa se creó y llenó un tablero Scrum en la url: https://trello.com/b/kQPOICCP/webphishing-dev. Este tablero se empezó a utilizar una vez se crearon los elementos básicos así como el backlog de mantenimiento lleno con las tareas que se solventaron a través del desarrollo.

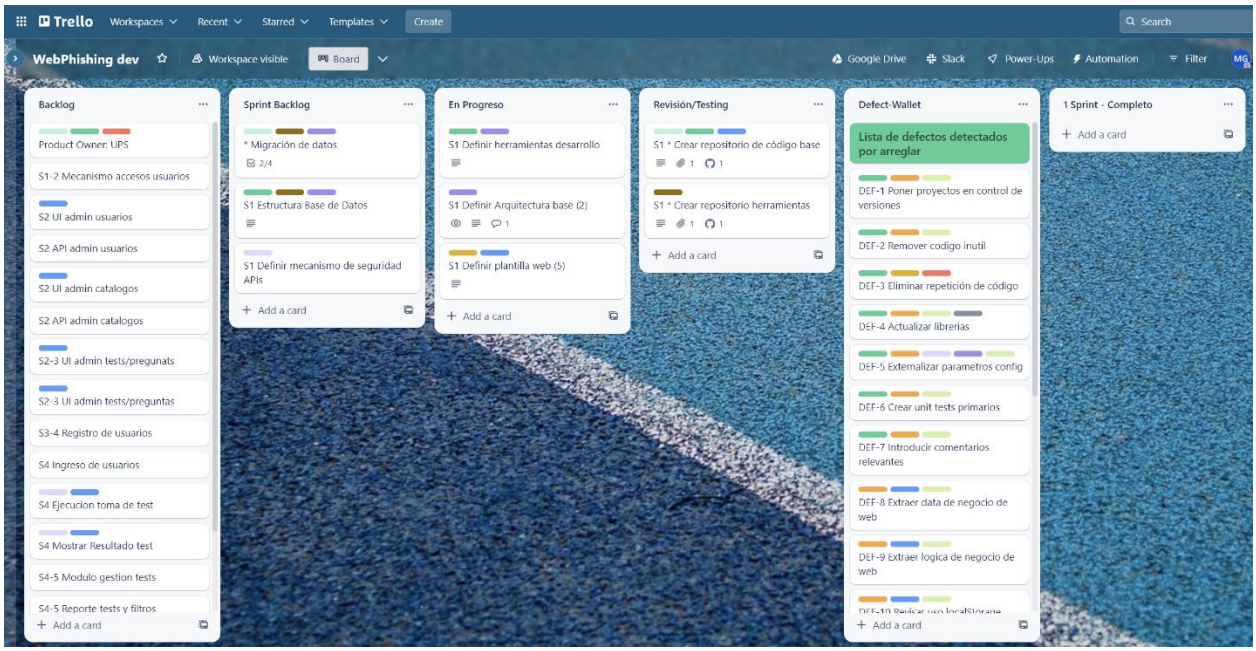

Ilustración 10. Tablero Scrum Inicial

Se crearon dos repositorios de código en el sitio Github, utilizando el espacio propio del autor mientras duró el desarrollo. Se creó un repositorio para el sistema de entrenamiento de phishing y otro para las herramientas adicionales que se utilizaron, principalmente en tareas de migración y mantenimiento.

Estos repositorios se trabajaron cada uno con dos ramas específicas:

- “main”: Rama estable del código al cual se utiliza para producción.
- “dev”: Para desarrollo continuo el cual se integró a main una vez realizadas las pruebas de validación y certificación.

Los enlaces a los repositorios son los siguientes:

- Sistema de Entrenamiento de Detección de Phishing: https://github.com/Vegeta/webphishing
- Herramientas adicionales de mantenimiento: https://github.com/Vegeta/webphishing-tools

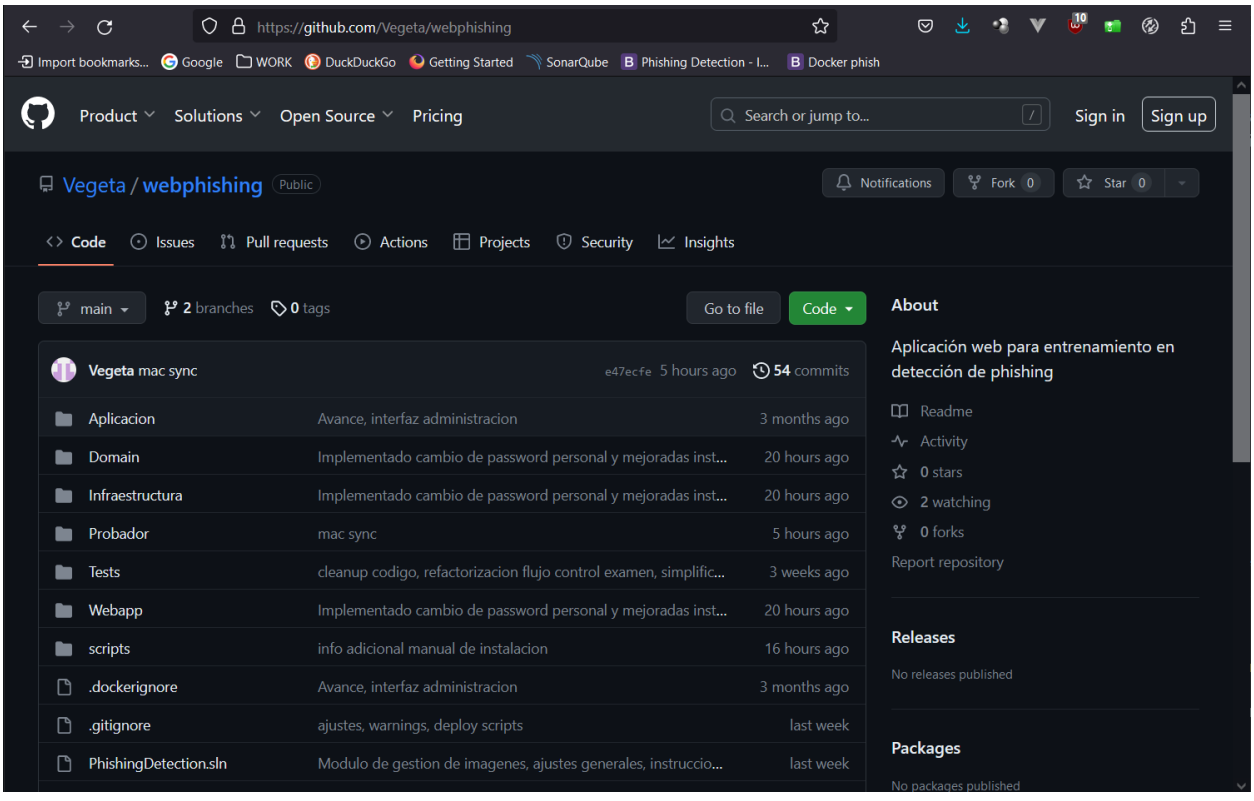

Ilustración 11. Repositorio en Github

Arquitectura

Una vez realizada la revisión del prototipo y definido el lineamiento general para el desarrollo del proyecto se inició con la propuesta de la arquitectura general del sistema considerando los recursos existentes al igual que el personal y el tiempo planificado. Se determinó que la opción más adecuada es crear una sola aplicación web separada internamente por distintas capas tratando de respetar principios de separación de responsabilidades y organizando los componentes por su función lógica enmarcados dentro de las convenciones de la tecnología escogida para el proyecto.

No se utilizará un modelo SPA para el frontend ya que es posible crear un nivel alto de interactividad usando el framework Vue.js directamente en páginas web tradicionales y también se puede utilizar componentes visuales sin necesidad de una aplicación independiente. De la misma forma, se descartó el uso de microservicios ya que el alcance del proyecto no requiere de una separación de componentes físicos que puedan funcionar y crecer de forma independiente.

Para la organización interna, se tomaron en cuenta varios patrones arquitectónicos muy difundidos y utilizados basados en los conceptos de Diseño Orientado al Dominio (Evans, 2003) y trabajos relacionados a aplicaciones empresariales como Patterns of Enterprise Application Architecture (Fowler, 2002) así como propuestas más actuales de estos temas por otros autores. Con esto en mente, se revisaron varias implementaciones de referencia de esta arquitectura, así como proyectos previos del autor con lo que se armó el esqueleto de la aplicación sobre el cual se construyó el resto de la funcionalidad específica. Esta arquitectura es flexible y se fue refinando durante el proyecto, incorporando nueva funcionalidad y mejorando código inicial en los siguientes sprints.

Adicionalmente, las entidades de negocio persistentes tienen una relación casi uno a uno con las estructuras de la base de datos lo cual facilita la comprensión del código, integración con herramientas y el mantenimiento del sistema a largo plazo.

Estructuras de datos

Con respecto al storage primario del sistema, se creó un esquema de base de datos relacional compatible con la información existente en el prototipo y enriquecido con información adicional como tablas de parametrización y auditoría, de acuerdo con lo especificado en los requerimientos. El esquema de la base de datos se muestra a continuación.

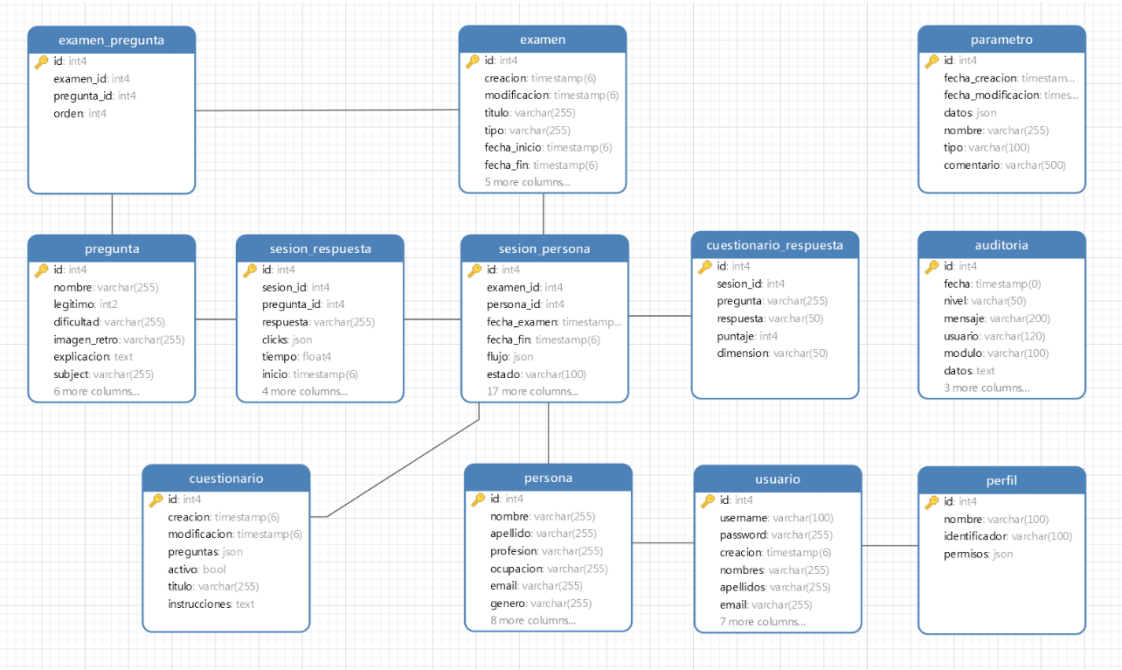

Ilustración 12. Diagrama Entidad Relación

Como datos relevantes, se destaca el uso de campos tipo JSON en varias tablas de la base de datos los que pueden contener datos arbitrarios y son utilizados para guardar estructuras complejas o listas que se observa pueden funcionar sin tablas relacionales. En la experiencia del autor, estas estructuras proporcionan gran versatilidad y presentan un buen balance entre un modelo de datos relacional y documental si se utilizan correctamente.

Para el almacenamiento de archivos cargados, en esta etapa inicialmente se consideró una carpeta en el disco del servidor de la aplicación configurada de forma simple con las rutas de los archivos físicos asociadas a registros en la base de datos.

Este esquema es la versión inicial creada para el primer sprint y durante el desarrollo se cambiaron campos y tablas para adecuarse a la funcionalidad, corregir errores detectados o apoyar a reportería (por ejemplo, con vistas).

Entidades de negocio

El diseño de las estructuras de datos se basa en entidades principales asociadas a entidades secundarias para mantener consistencia en los datos. A continuación, un resumen del rol de las entidades más importantes del sistema y sus relaciones:

- **Pregunta**: Guarda información de cada ejercicio de detección de phishing. Ya que esta entidad es la base de la información de resultados, se diseñó con borrado lógico a través del campo activo lo que previene la pérdida de datos si ya no se va a utilizar.
- **Examen**: Encabezado de la definición de un examen de phishing que contiene un grupo de preguntas a través de una relación muchos a muchos que puede tener opcionalmente un orden de las preguntas si no es automático.
- **Persona**: Contiene la información del perfil personal y profesional de la persona que interviene en los exámenes. Puede opcionalmente estar asociado a un usuario registrado.
- **Usuario y Perfil**: Estructuras para control de acceso al sistema por el proceso de autenticación y autorización. Contienen los datos de ingreso de los usuarios y los permisos de cada perfil respectivamente. Un usuario puede tener un solo perfil en esta versión y no necesita tener una persona a no ser que vaya a participar en exámenes.
- **Sesión persona**: Entidad principal que contiene la información del desarrollo de una prueba de phishing. Contiene los datos de control del flujo del examen y sincronización con la interfaz de usuario. La sesión está asociada a una persona. Adicionalmente tiene las respuestas a un cuestionario si este existe. La sesión mantiene una asociación con examen pero no es restrictiva.
- **Sesión respuesta**: Contiene las respuestas del usuario de la sesión personal así como el tiempo de respuesta e información adicional como los clics que hizo en la interfaz. Se asocia directamente con pregunta lo que minimiza los joins en consultas de reportes y

permite hacer consultas independientes así el examen base haya sido eliminado o cambiado.

- **Cuestionario respuesta**: Para efectos de reportería y control se almacena cada respuesta al cuestionario asociado a una sesión personal.
- **Parámetro**: Contiene registros de catálogos del sistema almacenados en campos JSON con estructura flexible. Tabla utilitaria, no tiene asociaciones de negocio.
- **Cuestionario**: Similar a la tabla parámetros, pero en una estructura separada que contiene la estructura de cuestionarios opcionales para los exámenes de phishing.
- **Auditoría**: Contiene un registro de las acciones importantes de negocio del sistema en un formato compatible con herramientas de log con un tipo de evento, fechas e información adicional opcional además de registro de usuario y dirección IP. Tabla utilitaria, no tiene asociaciones de negocio.

Convenciones de Base de datos

Para la consistencia y uniformidad de las estructuras en la base de datos se utilizaron las siguientes convenciones:

- Todos los nombres de objetos (tablas, campos, etc.) van en minúsculas y usando snake case (palabras separadas por guion bajo).
- Los nombres de las tablas van en singular.
- Toda tabla tiene un solo campo de clave primaria llamado “id”, de tipo entero y autogenerado.
- Las claves foráneas llevan el nombre de la entidad relacionada seguido del sufijo “_id”.
- Para PostgreSQL, la generación de las claves primarias se hace con el uso de secuencias automáticas.

Estas convenciones se apoyan en los principios de simplificación de datos definidos en (Evans, 2003) (Fowler, 2002), en particular la clave primaria única llamada id es utilizada por defecto por la mayoría de las bibliotecas de manejo de base de datos, en este caso Entity Framework.

Infraestructura inicial

El proyecto inicial se creó en una carpeta estructurada como una “Solución” de .NET la cual es el punto de entrada a varios proyectos de tipos distintos en sus propias carpetas. A cada uno de los proyectos creados se adicionaron las bibliotecas requeridas utilizando el gestor de paquetes Nuget.

Se creó el proyecto inicial de tipo Web, la cual es adecuado para su uso en aplicaciones web basadas en páginas simples, modelo MVC, servicios REST y otros servicios orientados a internet. En esta etapa no se consideró el esquema completo de seguridad el cual fue incorporado de forma gradual.

Adicionalmente, se definió la plantilla gráfica del sitio utilizando como base la biblioteca Bootstrap y JQuery, de la misma forma que en el prototipo, pero con las versiones más actualizadas. Como se definió en las herramientas del proyecto, se incluyó la última versión de Vue.js para proporcionar funcionalidad avanzada de interfaz de usuario en sistema. Para la parte administrativa del sitio, se utilizó otra plantilla gratuita y de libre distribución más adecuada para este tipo de funcionalidad.

Finalmente, para la parte de pruebas se creó un proyecto de código en la Solución donde se crearon las pruebas unitarias necesarias el cual puede ser ejecutado de forma independiente para ser incluido, por ejemplo, en un ciclo CI/CD o durante el mismo desarrollo.

Defectos resueltos: 1. Control de versiones, 2. Remover código inútil, 4. Actualizar las librerías, 9. Extraer la data de preguntas a una base de datos, 11. Eliminar estilos repetidos, 17. Revisar modelo de datos.

Detalle mantenimiento:

La creación del proyecto inicial con nuevas librerías resuelve el problema 4, mientras que los nuevos repositorios de código resuelven el problema 1. Sobre el código inútil encontrado en el prototipo, correspondiente al problema número 2, este aparentemente era producto de

generadores de código que principalmente crean clases con métodos importantes vacíos, listos para que el desarrollador los complete o los borre. En la nueva solución, se revisó que el código generado (por ejemplo, migraciones de base de datos) no contenga estos métodos si no son necesarios.

Con el uso de las plantillas web para el sitio, se está atacando el problema 11 ya que inicialmente, se utilizaron los estilos CSS predeterminados para los elementos comunes y más adelante, las personalizaciones fueron centralizadas en otro archivo específico llamado site.css que incluye estilos nuevos y sobreescrituras utilizando selectores DOM para el HTML. En el caso de componentes que requieren estilos muy específicos solo para ciertos elementos (como las plantillas HTML de las preguntas, por ejemplo), se trató de utilizar estilos in-line en su mayoría u hojas personalizadas, pero siempre con un selector apuntando al elemento DOM particular.

Al tratarse de un nuevo sistema web, la solución a este problema es casi automática sin embargo si se quiere resolver en un sistema existente, la solución propuesta debería ser similar, es decir, centralizar todos los estilos en archivos CSS principales y evitar utilizar estilos por cada página o por cada componente si se trata de un framework SPA.

Por otro lado, el problema número 17 se resuelve al crear una nueva base de datos que incluye una separación más clara de las estructuras de datos, los tipos de datos de los campos y sus relaciones y que, adicionalmente, es más fácil visualizar utilizando herramientas de gestión de base de datos como la que se usó para el presente proyecto.

Migración de datos de preguntas

Uno de los problemas más relevantes observados durante el análisis del prototipo fue el uso de código de frontend como base de datos para las preguntas del módulo de examen de phishing, la cual corresponde a la tarea 9 del backlog de mantenimiento. La resolución de este problema involucró los siguientes pasos:

- Extracción de los datos desde archivos HTML y Typescript.
- Transformación y procesamiento intermedio.
- Carga de la información en nueva base de datos.
- Limpieza de datos.

Estas tareas se realizaron con una plataforma adicional basada en PHP la cual fue creada específicamente para este proyecto.

Se realizó la extracción y separación de este contenido con técnicas de procesamiento de HTML en servidor, similar a screen scrapping, y se pudo obtener una base de datos de las preguntas con los elementos preparados para uso y edición. Cabe resaltar que no fue posible limpiar el 100% del código HTML debido a que la fuente original presentaba inconsistencias y algunas preguntas tuvieron que ser editadas a mano con la nueva herramienta.

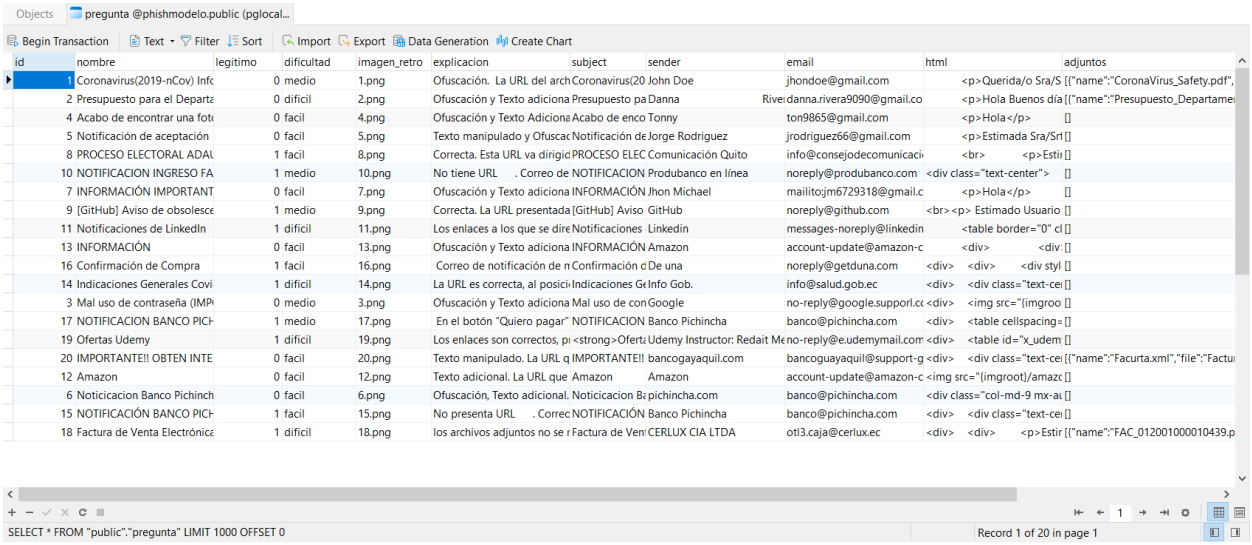

Ilustración 13. Preguntas Migradas A La Nueva Base De Datos

4.2.2. SPRINT 2

Backlog: 3. Infraestructura inicial, 4. Administración de usuarios, 5. Administración de catálogos.

Parametrización y seguridad

Continuando con la refinación de la infraestructura, se crearon nuevas clases para temas generales de seguridad y auditoría lo cuales se utilizan a través de todo el sistema. Para este caso, la seguridad involucra el control de acceso a las páginas a ciertas partes del módulo de cliente final (quien toma los exámenes) y al módulo de administración.

Se creó un subsistema simple que relaciona las entidades usuario con perfiles, que a su vez tienen permisos de operaciones que pueden realizar en el sistema. Existe un super usuario administrador que tiene capacidad completa de administración. Esta información se guarda en la base de datos en las tablas definidas en el sprint anterior.

El control de acceso a los módulos se realizó de forma semiautomática utilizando atributos de autorización incorporados en la plataforma para clases y métodos de controladores y otras clases específicas que contienen los permisos por funcionalidad para control de la interfaz de usuario y procesos. En el caso del acceso a la evaluación de phishing, se almacenan y utilizan datos de la sesión del usuario actual conjuntamente con las definiciones de permisos en código y base para permitir acceder a la evaluación una vez se ha pasado por el registro inicial.

Para administración de usuarios y perfiles, se crearon módulos de gestión que constan de una pantalla de lista con filtros y una pantalla de vista/edición que permite cambiar los datos de cada registro. En el caso de usuarios se incluye información básica y datos operativos como el perfil y si está habilitado o no. Para el caso de perfiles, se creó una estructura interna de los posibles permisos que puede contener cada perfil y en la interfaz de usuario, una lista simple con cajas de check que permite marcar que permisos va a tener el perfil.

Dentro de la parametrización del sistema, se utilizó inicialmente el mecanismo por defecto de .NET Core que utiliza un archivo de configuración JSON externo y se adicionó el uso de variables

de entorno del sistema operativo de forma opcional para leer ciertos parámetros de sistema como la cadena de conexión a la base de datos, el tipo de entorno (desarrollo, producción) y otros. Las variables de entorno, en particular, son muy utilizadas en despliegues sobre contenedores Docker y es la forma por defecto de pasar parámetros de configuración a la aplicación.

Se creó un módulo genérico para edición de variables de negocio el cual, en esta etapa, permite la creación de catálogos de tipo clave valor que se pueden utilizar en varias partes del sistema. En esta etapa, la información de catálogos se guardó en la tabla “parametro” que tiene un identificador del tipo de parámetros del registro y un campo de tipo JSON para los datos operativos concretos. En este esquema se obtiene mucha flexibilidad para la clase de información que se quiera parametrizar, sin embargo, es responsabilidad del desarrollador convertir la información del parámetro en los tipos concretos necesarios usando deserialización JSON.

Con respecto a la infraestructura de comunicaciones entre backend y frontend, se determinó que no existirán llamadas a APIs externas en esta versión por lo que la seguridad CORS detallada no es necesaria ya que los componentes de interfaz de usuario hacen solo llamadas locales a endpoints en controladores dentro del mismo dominio. La plataforma proporciona opciones fáciles para configurar esta seguridad en clases y métodos que exponen servicios si fuera necesario.

Defectos resueltos: 5. Parametrización de variables, 14. Revisar seguridad CORS

Detalle mantenimiento:

Desde la creación del esqueleto del proyecto, la plataforma utiliza variables externas para la configuración de arranque del sistema lo cual fue complementado con otras formas de obtener esta información. Adicionalmente se crearon mecanismos para guardar datos arbitrarios de configuración de datos de negocio del sistema lo cual resuelve el problema número 5. Para resolver esto en un sistema existente, sería necesario revisar qué mecanismos proporciona la plataforma y cambiar las variables quemadas por llamadas a métodos de configuración,

dependiendo de la plataforma. Los parámetros de negocio del sistema pueden ser implementados de varias formas distintas, siempre y cuando exista una parametrización dinámica, por lo que no existe una forma única de llegar a una solución.

La seguridad es un aspecto general de cualquier sistema y está presente en varias partes de la aplicación por lo que los problemas encontrados en el prototipo fueron resueltos a lo largo de varios sprints, comenzando por este en el cual se crearon las bases. Concretamente se definió la política CORS para acceso a los APIs los cuales, en esta arquitectura del proyecto, solo serán consumidos internamente como ya se mencionó, sin embargo, la configuración es explícita y permite cambiarla posteriormente de ser necesario.

Existen varias opciones para implementar cierto nivel de control de acceso en aplicaciones web, incluyendo mecanismos nativos como Identity para .NET por lo que esto podría ser cambiado o mejorado en una futura revisión sin afectar demasiado a la base de código actual.

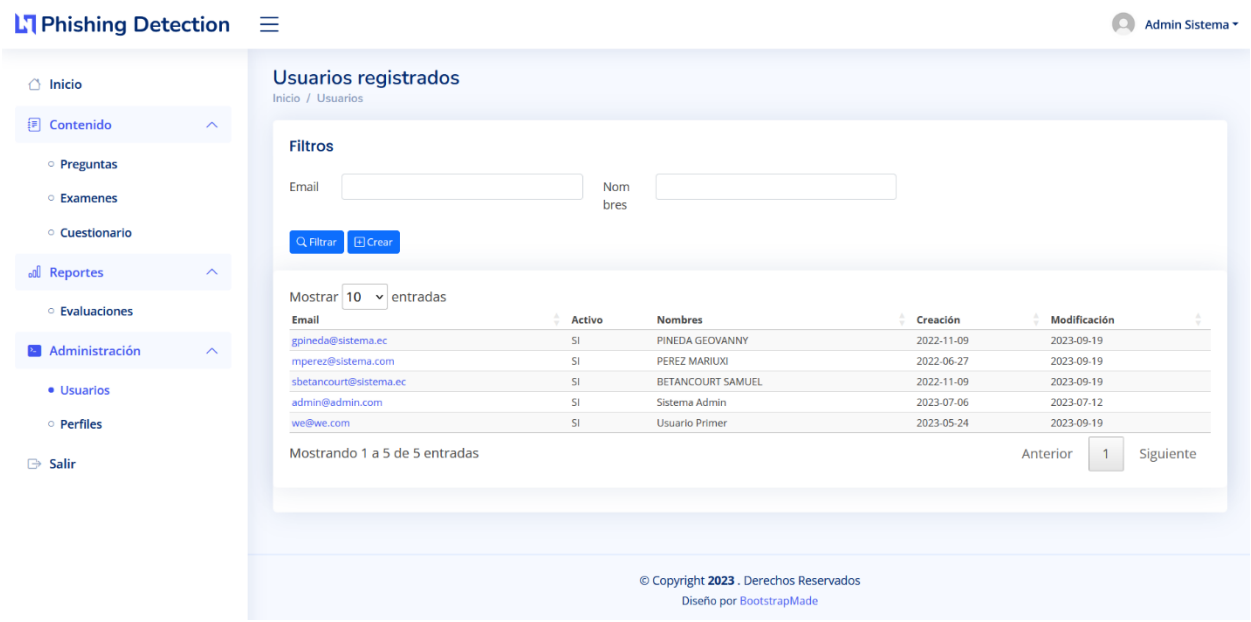

Ilustración 14. Módulo De Administración

4.2.3. Sprint 3

Backlog: 6. Administración de tests y preguntas, 7. Ingreso de usuario registrado.

Detalle y actividades:

Para esta versión, se decidió mantener el esquema simple de ingreso de usuarios externos para que puedan tomar tests de phishing, esto es, a través de un formulario de registro de datos que habilita al módulo de exámenes. Se mejoró la interfaz de usuario del formulario y se incluyeron validaciones a los datos utilizando componentes Javascript comunes como jQuery validation plugin (https://jqueryvalidation.org/).

Por otra parte, se crearon pantallas para la administración de las preguntas de los exámenes utilizando un esquema de edición de código HTML en pantalla con previsualización y carga de imágenes. Se utilizó la librería CodeMirror para el editor y la funcionalidad básica de carga de archivos del sistema (https://codemirror.net).

Con este componente fue posible editar algunas de las preguntas migradas y arreglar el código que no se pasó correctamente. Se incorporó un subsistema para reconocer hipervínculos en contenido de las preguntas (que son básicamente mensajes de email) que permite automatizar la detección de cuando un usuario pasa el ratón por el hipervínculo, funcionalidad que se encontraba en el prototipo.

Se implementó un módulo para la creación de exámenes de phishing, los cuales tienen una identificación, cierta configuración de comportamiento y una lista de ejercicios asociados. Este módulo permite crear varios exámenes que pueden ser utilizados como la plantilla general por defecto o, en un futuro, para evaluaciones específicas si así lo decide el administrador de contenido.

Adicionalmente, se creó un registro para crear cuestionarios generales que se incluyen en los exámenes de phishing y se usan como complemento a la evaluación y para estadísticas.

Defectos resueltos: 5. Parametrización de variables

Detalle mantenimiento:

La creación de los módulos de administración sirve para resolver el problema número 5 sobre parametrización de variables. La nueva funcionalidad de control de usuarios complementa también a la solución de los problemas de seguridad con un mecanismo más robusto de autenticación en la plataforma.

De la misma forma, para una aplicación existente que presente estos problemas la solución es similar, cambiar las variables quemadas por código dinámico utilizando las prestaciones de la plataforma o bibliotecas adicionales.

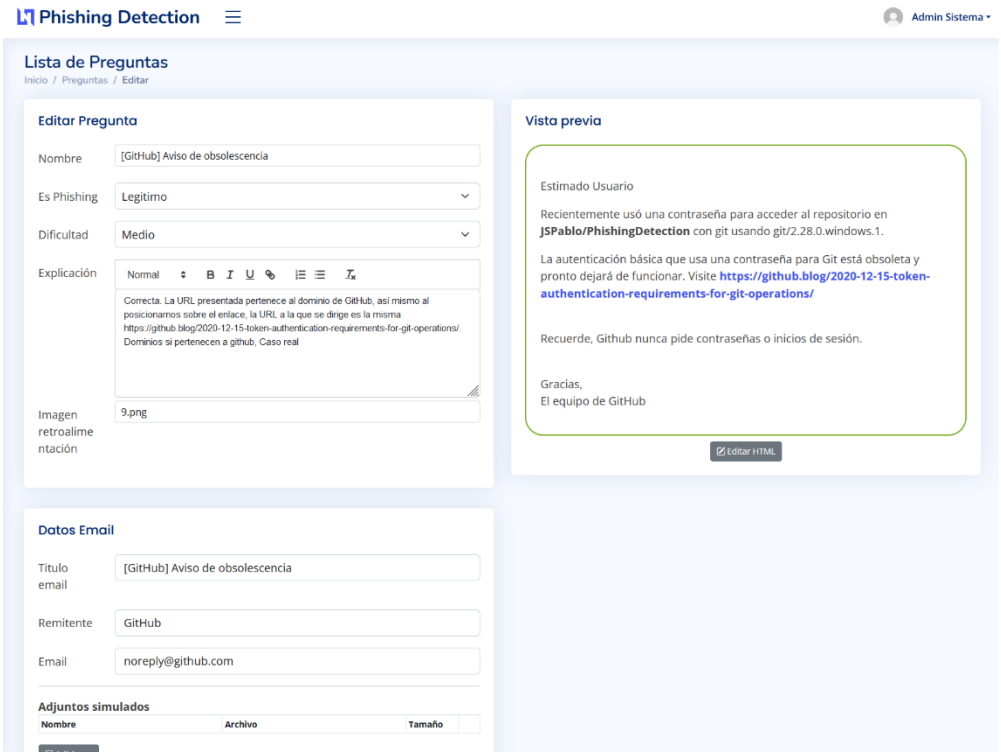

Ilustración 15. Edición Y Creación De Preguntas

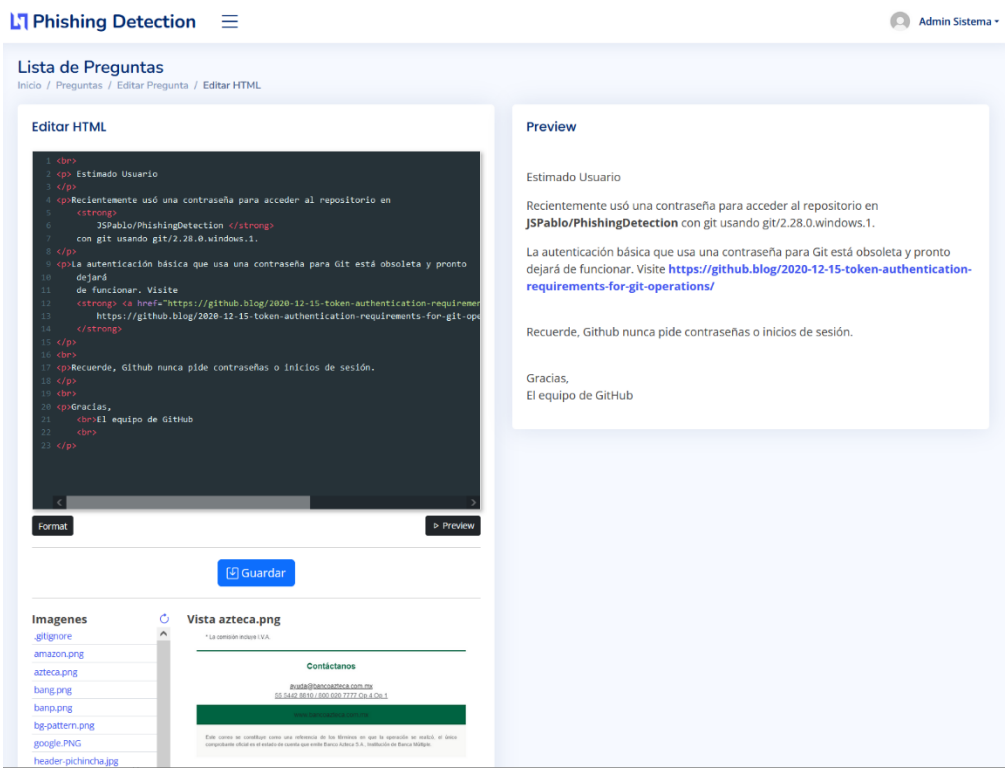

Ilustración 16. Editor Html De Mensaje De Email

4.2.4. Sprint 4

Backlog: 8. Ingreso de usuario registrado, 9. Módulo de toma de exámenes.

Detalle y actividades:

Se creó una pantalla de acceso para usuarios registrados que permite acceder a los módulos de gestión de contenido y reportes de la plataforma utilizando un mecanismo de filtro en los controladores marcados como seguros. Este filtro se encarga de redirigir al navegador a una pantalla de ingreso de credenciales si no se encuentra información de usuarios en la sesión actual.

Adicionalmente, se crearon las primeras pantallas de toma de exámenes instrumentadas con un control de flujo que permite el acceso al usuario registrado así como retomar un examen en curso en el punto donde se quedó si se cierra en navegador o se sale de la página por cualquier motivo.

Con esta infraestructura y la parametrización en estado funcional, se creó el módulo para responder a las preguntas de un examen de phishing. Este módulo contiene un diseño similar al utilizado en el prototipo el cual se asemeja a un cliente de correo electrónico web que muestra mensajes de email. Una vez en el examen, el usuario debe determinar si el mensaje que está viendo es legítimo o es phishing para lo cual existen dos botones con estas opciones y una ventana de justificación de la respuesta escogida. Cada mensaje puede tener hipervínculos simulados que ayudan al usuario en su respuesta al ejercicio y además se usan en la evaluación para determinar si la persona utilizó estos elementos en su decisión y luego para estadísticas internas.

En el aspecto técnico, el cambio más importante con respecto al prototipo es la lectura de las preguntas de forma dinámica desde la base de datos y el cambio en el control del flujo, pasando más responsabilidades al backend el cual lleva ahora todo el registro del examen y deja al frontend con la responsabilidad de presentación de datos al usuario. Se decidió no utilizar el componente localStorage y en su lugar se guarda solo un token de sincronización en la sesión de usuario, la cual se mantiene en el backend y se sincroniza automáticamente utilizando cookies sin exponer información sensible al frontend lo cual mejora el perfil de seguridad del sistema.

Por el lado del servidor, la lógica de decisión de preguntas y el cálculo de resultados se encuentra ahora en esta capa para lo cual se implementaron servicios de negocio a cargo de esta y otras funcionalidades complementarias.

Finalmente, se implementó la pantalla de resultados de forma similar al prototipo pero con mayor funcionalidad como imprimir el reporte y ver la explicación de cada pregunta de mejor forma utilizando navegación dentro de la página.

Defectos resueltos: 3. Eliminar repetición de código, 8. Extraer lógica de negocio, 10. Minimizar el uso de localStorage, 12. Implementar la lógica de negocio, 13. Reprogramar los algoritmos principales, 15. Implementar APIs, 16. Implementar seguridad APIs

Detalle mantenimiento:

En este sprint se resolvieron los problemas principales relacionados con la lógica de negocio del sistema, la cual estaba implementada en la capa de frontend en el prototipo y se encontró algunas anomalías como falta de claridad en los algoritmos y repetición de código.

Esta reimplementación se utilizó para construir APIs de operaciones de negocio que dan la funcionalidad principal al módulo de toma de exámenes de phishing que se puede considerar el corazón del sistema. Este trabajo sirve para solventar los problemas 3, 8, 12, 13 y 15 del backlog de mantenimiento.

La implementación de la nueva interfaz para toma de exámenes utiliza un mecanismo más simple para el flujo de las preguntas, basado en el consumo de los endpoints internos minimizando la información guardada en browser, lo que resuelve el problema 10.

La mejora en la forma de ingresar al módulo de exámenes y su manejo complementa las soluciones de seguridad relacionadas al problema 16 al utilizar los APIs con un token de sincronización y solo desde las páginas de la aplicación sin exposición externa.

Ilustración 17. Nueva Pantalla Inicial

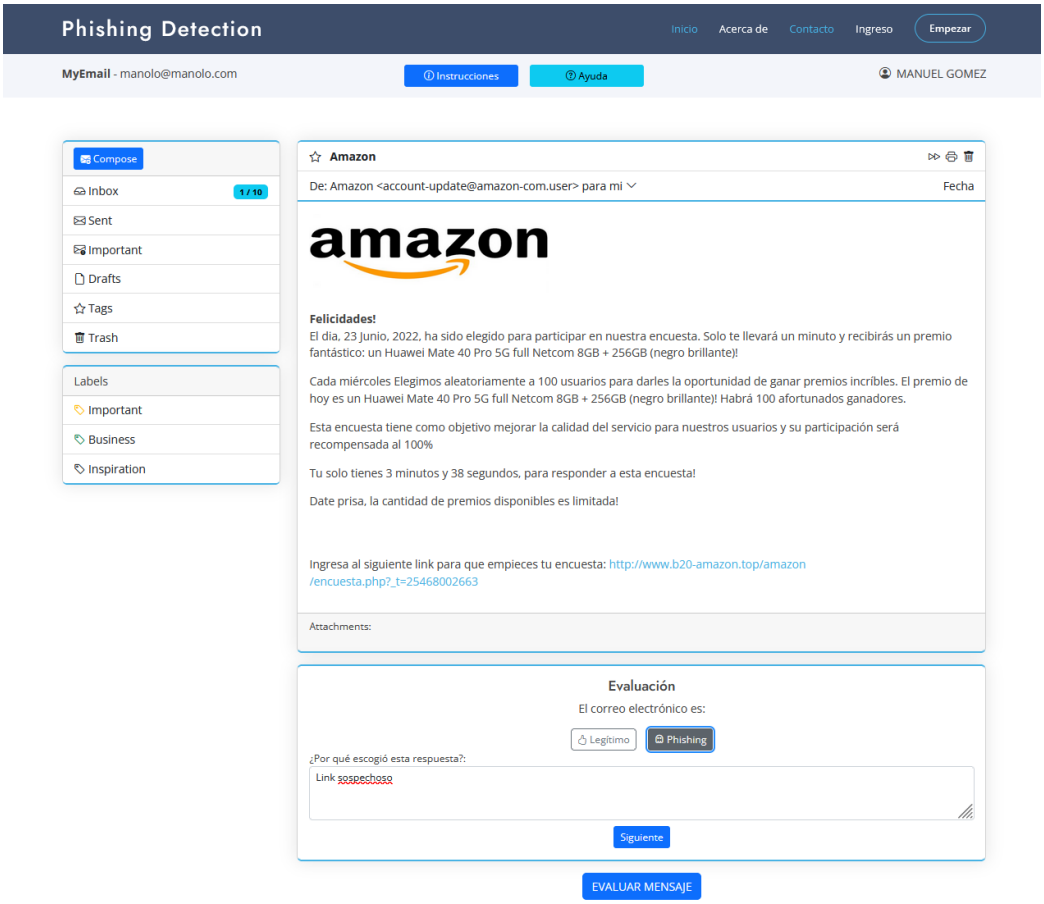

Ilustración 18. Evaluación De Pregunta De Phishing

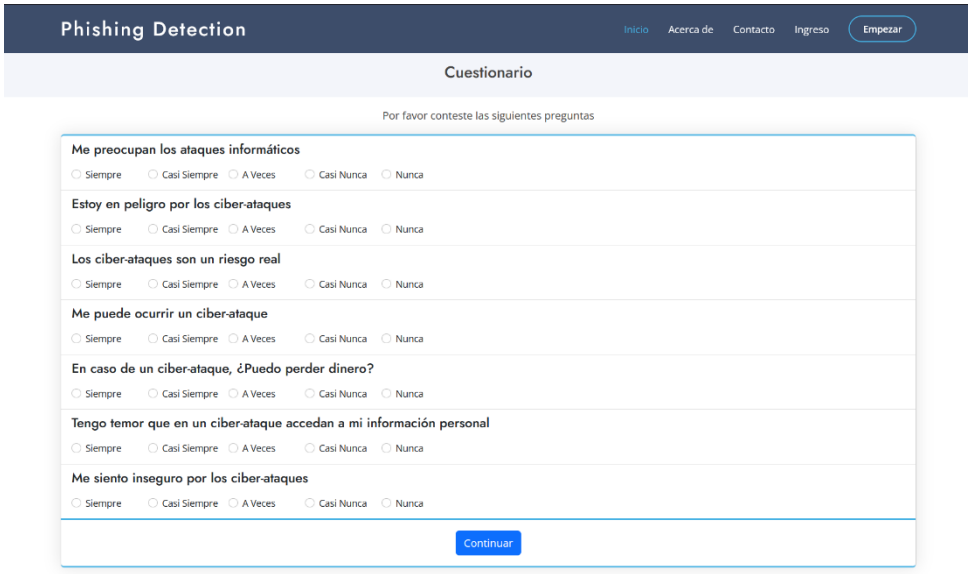

Ilustración 19. Cuestionario De Percepción De Seguridad

4.2.5. SPRINT 5

Backlog: 11. Reporte de resultados de exámenes, 12. Integración y despliegue, 13. documentación de despliegue (nueva tarea), 14. Pruebas de despliegue (nueva tarea).

La funcionalidad para gestión de sesiones ya fue completada y probada en sprints anteriores por lo que no este ítem no se incluyó en el backlog de este sprint y pasó a completado en el sprint número 4. Se adicionaron dos ítems en el backlog del producto que fueron ejecutados en este sprint y corresponden a realizar pruebas de despliegue en varias plataformas y a generar documentación final para el proyecto.

Reportería inicial

Se creó un módulo para consultar los resultados de exámenes realizados que permite visualizar los datos tal como los mira el usuario final al terminar la prueba. Este módulo posee filtros de fechas, calificaciones y datos de perfil; se incluye además la capacidad de exportar las listas filtradas a una hoja de cálculo en formato Excel .xlsx.

Adicionalmente se creó una pantalla para visualización de datos agregados al estilo Dashboard permite ver un resumen de las pruebas y las preguntas. Para esta etapa se crearon dos reportes principales para facilitar a los skateholder la obtención de datos que permitan realizar estudios con más profundidad de forma externa. Sin embargo, posteriormente se podría incorporar más funcionalidad de inteligencia de negocio en el sistema de ser requerido.

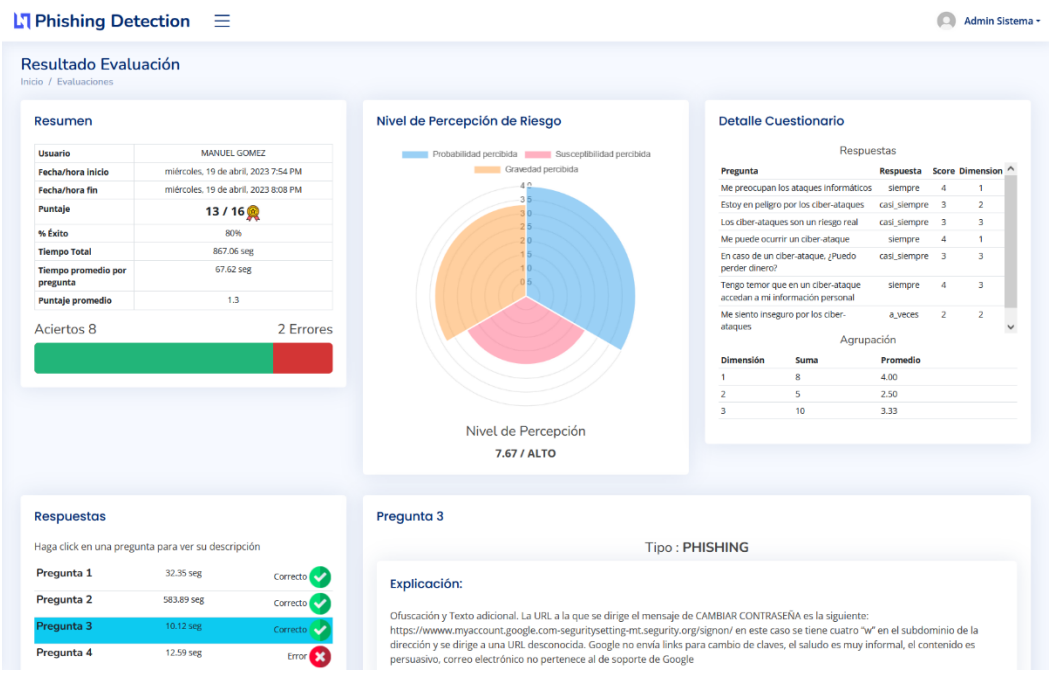

Ilustración 20. Reporte De Resultados De Evaluación

Pruebas de despliegue

Previo a la instalación en el ambiente de producción, se realizaron varias pruebas de despliegue en distintos ambientes con el fin de asegurar que el sistema sea portable entre varias plataformas. Los ambientes donde se probó la instalación del sistema fueron:

- Internet Information Server 10 sobre Windows 10.
- Servidor Kestrel nativo .NET Core + proxy NGINX como frente. Probado en Windows, MacOS y Linux.
- Contenedor Docker usando archivos incluidos en la Solución.
- Servicio EC2 de Amazon Web Services, máquinas tipo micro y small con Amazon Linux 2.

La configuración de cada ambiente se hizo de forma estándar siguiendo las recomendaciones de cada fabricante o proveedor. El servidor Kestrel que viene incluido en la plataforma soporta el uso de certificados digitales para comunicación SSL incluso durante el desarrollo por lo que es posible activar el acceso por protocolo seguro usando esta solución o, configurar los certificados directamente en los servidores web del alojamiento final si así se requiere.

Despliegue y entrega

De acuerdo a definiciones previas, la aplicación se instaló sobre infraestructura proporcionada por la Universidad Politécnica Salesiana.

Con el fin de minimizar costos de licencias y servicios, se decidió utilizar como plataforma base Linux, concretamente la distribución AlmaLinux 9 (https://almalinux.org/), un sistema moderno compatible con Red Hat Enterprise, CentOS y otras distribuciones ampliamente utilizadas y con soporte disponible.

Se realizó una solicitud de recursos informáticos al personal de Data Center de la Universidad y se asignó una máquina virtual con 2 CPUs virtuales, 4GB de RAM y 80GB de disco duro preinstalada con el sistema operativo base y accesible remotamente a través del software de escritorio remoto AnyDesk (https://anydesk.com/en).

Por otra parte, se decidió automatizar la creación y despliegue de servicios utilizando contenedores Docker (https://www.docker.com/) con el fin de facilitar la implementación del sistema sobre cualquier otra infraestructura que lo soporte si se genera la necesidad. Para esta configuración se crearon dos archivos principales:

- Dockerfile: Incluido en el proyecto de aplicación web y creado desde la misma solución .NET. Contiene los comandos para la compilación y ejecución de la aplicación de forma automática. El proceso de creación del contenedor utiliza dos imágenes:
    - mcr.microsoft.com/dotnet/sdk:7.0 para compilación.
    - mcr.microsoft.com/dotnet/aspnet:7.0 para ejecución.
- docker-compose.yml: Definición de servicios, exposición de puertos y volúmenes de datos. Para este proyecto se crearon dos servicios, la aplicación web que utiliza el archivo Dockerfile y un contenedor para la base de datos. Para este último se utilizó una de las últimas imágenes estables de postgresql disponibles al momento en una versión ligera: postgres:15-alpine.

Para el despliegue concreto del sistema se realizaron las siguientes tareas:

- Verificación de la plataforma base, seguridad existente y disponibilidad del gestor de paquetes dnf.
- Instalación y configuración de servicios Docker.
- Copia de código del sistema al servidor lista para compilación.
- Copia de backup de base de datos al servidor.
- Configuración de carpetas para la base de datos: pgdata y backups.
- Despliegue automatizado de los servicios utilizando el comando “docker compose up -d”.
- Restauración del backup de la base de datos.
- Verificación de despliegue y funcionamiento.

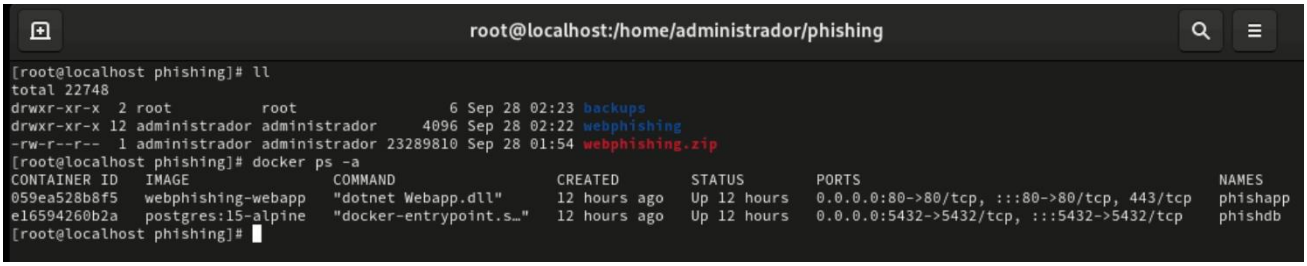

Ilustración 21. Aplicación desplegada en producción con Docker

Como último paso del proyecto, se generaron entregables que sirven para facilitar el despliegue en la mayoría de los ambientes posibles para el alojamiento de la aplicación y para el desarrollo sobre el sistema:

- Respaldo de base de datos en formato binario PostgreSQL con la data migrada de la versión anterior y acceso de administración configurado.
- Scripts DDL de generación de la base de datos.
- Respaldo de la carpeta de la Solución con todo el código, insumos adicionales y control de versiones en formato zip. No se incluye la descarga de librerías externas.
- Respaldo de la carpeta con el código complementario para la migración a la nueva plataforma incluyendo control de versiones en formato zip. No se incluye la descarga de librerías externas.

- Scripts de despliegue para plataforma Docker, utilizado durante pruebas y para generar la plataforma de producción.

En el proyecto de código, se incluyó un manual para la instalación del sistema en formato Markdown (archivo .md) el cual puede ser visualizado directamente en la web de github o se puede editar como texto.

Defectos resueltos: 18. Verificación y pruebas de cambio

Detalle mantenimiento:

Adicional a las pruebas automáticas y manuales realizadas durante el desarrollo, en esta última etapa se realizó una validación de la funcionalidad requerida con el fin de asegurar el cumplimiento de los requerimientos. Se encontró que la aplicación cumple con los parámetros mínimos para ser instalada en un ambiente de producción y comenzar con su uso como se hizo originalmente con el prototipo que originó la nueva plataforma. Esta definición resuelve el ítem número 18 del backlog de mantenimiento.

5. Resultados y Discusión

En el estudio de refactorización presentado, se aborda uno de los desafíos fundamentales que enfrentan las organizaciones al modernizar sus sistemas de software existentes, la falta de una metodología clara para iniciar proyectos de refactorización. Las posibles anomalías encontradas y los requerimientos de mejoras pueden incrementar la complejidad de las tareas de mantenimiento por lo que contar con un marco de referencia inicial es clave para el éxito del proyecto.

Una de las contribuciones principales de este estudio es la creación de una metodología sólida que aborda esta problemática de manera integral. Esta metodología se distingue por tres aspectos principales:

1. Contexto del Análisis de Código: En primer lugar, reconoce la importancia de comprender profundamente el contexto del código existente. Esto implica una revisión exhaustiva de la arquitectura, las dependencias, las tecnologías subyacentes y las limitaciones actuales del sistema. Esta comprensión completa es esencial para tomar decisiones informadas sobre qué partes del código deben refactorizarse y en qué medida.
2. Requerimientos de los Usuarios: Además, la metodología incorpora una evaluación continua de los requerimientos de los usuarios. Dado que estos pueden evolucionar con el tiempo, es esencial mantener una comunicación constante con los stakeholders y ajustar el enfoque de refactorización según sea necesario. Esto garantiza que el software refactorizado siga siendo relevante y útil para sus usuarios finales.
3. Herramientas de Análisis Automático: Por último, se destaca el uso de herramientas de análisis automático como parte de la metodología. Estas herramientas pueden ayudar a identificar áreas problemáticas en el código de manera eficiente, lo que permite a los desarrolladores enfocar sus esfuerzos en las áreas críticas. Además, las métricas generadas por estas herramientas brindan una base objetiva para la toma de decisiones.

En conjunto, esta metodología proporciona un marco de trabajo para abordar proyectos de refactorización de manera sistemática y eficiente. Al combinar un conocimiento más profundo del código existente, una comprensión continua de los requerimientos de los usuarios y el uso de herramientas de análisis automático, las organizaciones pueden minimizar los riesgos y maximizar los beneficios de sus proyectos de refactorización.

En resumen, este estudio muestra que el éxito en la refactorización de sistemas legados no solo depende de las habilidades técnicas, sino también de la adopción de una metodología efectiva que integre el contexto del código, las necesidades cambiantes de los usuarios y el análisis automatizado. Esta metodología puede servir como un valioso recurso para equipos de desarrollo que buscan mejorar la calidad y la mantenibilidad de sus plataformas de software.

6. Conclusiones

El estudio realizado sobre el Sistema de entrenamiento de detección de phishing permitió comprender varios desafíos que se presentan en proyectos de mantenimiento de software, en especial cuando se trata de analizar posibles refactorizaciones y existen mejoras propuestas.

En este estudio, se reconoció la carencia de un enfoque definido para iniciar proyectos de refactorización y se abordó este problema mediante la creación de una metodología simple que proporcionó una guía clara para realizar procesos de refactorización.

La decisión de reconstruir la aplicación en lugar de continuar con el mantenimiento tradicional indicó la gravedad de las fallas y limitaciones en el sistema original. Esto sugiere que, en algunos casos, la refactorización completa puede ser más beneficiosa y rentable que continuar con las correcciones incrementales.

La adopción de una metodología ágil, con un enfoque en el mantenimiento, demostró ser efectiva en este estudio. Colocar el mantenimiento como punto central permite una mayor flexibilidad y adaptabilidad a medida que se realizan cambios en el sistema.

La metodología simple creada en este estudio podría servir como punto de partida para otros proyectos de refactorización en situaciones similares. Proporciona una estructura clara para abordar problemas de mantenimiento o refactorización y puede adaptarse según las necesidades específicas del proyecto.

Es importante recordar que cada proyecto de refactorización es único, y las decisiones sobre si reconstruir o no la aplicación deben basarse en un análisis cuidadoso de las circunstancias individuales, incluyendo la complejidad del sistema, la gravedad de las fallas y las capacidades y recursos disponibles para el desarrollo.

En resumen, este estudio destaca la importancia de tener una metodología clara y adaptable para abordar proyectos de refactorización, así como la efectividad de una metodología ágil cuando se coloca el mantenimiento en el centro de la estrategia. También se enfatiza la necesidad de una evaluación cuidadosa antes de tomar decisiones importantes sobre el enfoque a seguir en proyectos de mantenimiento de software.

Referencias

Al-hamar, Y., & Kolivand, H. (2020). A New Email Phishing Training Website. 2020 13th International Conference on Developments in ESystems Engineering (DeSE), 263–268. https://doi.org/10.1109/DeSE51703.2020.9450238

Alkhaeir, T., & Walter, B. (2021). The Effect of Code Smells on the Relationship Between Design Patterns and Defects. IEEE Access, 9, 3360–3373. https://doi.org/10.1109/ACCESS.2020.3047870

Ashraf, S., & Shabib, A. (2017). IScrum: An Improved Scrum Process Model. International Journal of Modern Education and Computer Science, 9(8), 16–24. https://doi.org/10.5815/ijmecs.2017.08.03

Briceño, A., Anabel, D., Valarezo Bracho, D. I., & Fuertes Díaz, W. M. (2021). Plataforma Web para entrenamiento de ataques de Phishing mediante seguridad y psicología cognitiva. http://repositorio.espe.edu.ec/bitstream/21000/32745/1/T-ESPE-052520.pdf

Chathuranga, H. (2021). A Systematic Review of Code Smell Detection Approaches. https://doi.org/10.5281/ZENODO.4738772

de la Torre, C., Wagner, B., & Rousos, M. (2023, March 1). .NET Microservices. Architecture for Containerized .NET Applications. https://learn.microsoft.com/en-us/dotnet/architecture/microservices/

Desolda, G., Ferro, L., Marrella, A., Costabile, M., & Catarci, T. (2022). Human Factors in Phishing Attacks: A Systematic Literature Review. ACM Computing Surveys, 54, 35. https://doi.org/10.1145/3469886

Diaz, A., Sherman, A. T., & Joshi, A. (2018). Phishing in an Academic Community: A Study of User Susceptibility and Behavior (arXiv:1811.06078). arXiv. https://doi.org/10.48550/arXiv.1811.06078

Divakaran, D. M., & Oest, A. (2022). Phishing Detection Leveraging Machine Learning and Deep Learning: A Review. IEEE Security & Privacy, 2–11. https://doi.org/10.1109/MSEC.2022.3175225

Fowler, M. (2018). Refactoring: Improving the Design of Existing Code (2nd edition). Addison-Wesley Professional.

Golubev, Y., Kurbatova, Z., Alomar, E., Bryksin, T., & Mkaouer, M. W. (2021). One thousand and one stories: A large-scale survey of software refactoring (p. 1313). https://doi.org/10.1145/3468264.3473924

Han, X., Tahir, A., Liang, P., Counsell, S., & Luo, Y. (2021). Understanding Code Smell Detection via Code Review: A Study of the OpenStack Community. 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC), 323–334. https://doi.org/10.1109/ICPC52881.2021.00038

Hinderks, A. (2019). A Methodology for Integrating User Experience Methods and Techniques into Agile Software Development.

Hussain, R. G., & Javed, A. (2015). Qualitative Approach For Estimating the Influence Of Refactoring And Scrum In Software Development. 3(2), 9.

IEEE. (2018, November 30). 29148-2018—ISO/IEC/IEEE International Standard—Systems and software engineering—Life cycle processes—Requirements engineering. https://www.iso.org/standard/72089.html

Kasauli, R., Knauss, E., Horkoff, J., Liebel, G., & de Oliveira Neto, F. G. (2021). Requirements engineering challenges and practices in large-scale agile system development. Journal of Systems and Software, 172, 110851. https://doi.org/10.1016/j.jss.2020.110851

Kaur, A., Jain, S., Goel, S., & Dhiman, G. (2021). A Review on Machine-learning Based Code Smell Detection Techniques in Object-oriented Software System(s). Recent Advances in Electrical & Electronic Engineering (Formerly Recent Patents on Electrical & Electronic Engineering), 14(3), 290–303. https://doi.org/10.2174/2352096513999200922125839

Kaya, M., Conley, S., Othman, Z. S., & Varol, A. (2018). Effective software refactoring process. 2018 6th International Symposium on Digital Forensic and Security (ISDFS), 1–6. https://doi.org/10.1109/ISDFS.2018.8355350

Kerievsky, J. (2004). Refactoring to Patterns (1st edition). Addison-Wesley Professional.

Kim, M., Zimmermann, T., & Nagappan, N. (2012). A field study of refactoring challenges and benefits. Proceedings of the ACM SIGSOFT 20th International Symposium on the

Foundations of Software Engineering - FSE ’12, 1. https://doi.org/10.1145/2393596.2393655

Lacerda, G., Petrillo, F., Pimenta, M., & Gueheneuc, Y. G. (2020). Code Smells and Refactoring: A Tertiary Systematic Review of Challenges and Observations. https://doi.org/10.1016/j.jss.2020.110610

Martini, A., Fontana, F. A., Biaggi, A., & Roveda, R. (2018). Identifying and Prioritizing Architectural Debt Through Architectural Smells: A Case Study in a Large Software Company. In C. E. Cuesta, D. Garlan, & J. Pérez (Eds.), Software Architecture (Vol. 11048, pp. 320–335). Springer International Publishing. https://doi.org/10.1007/978-3-030-00761-4_21

Mohanani, R., Ralph, P., Turhan, B., & Mandić, V. (2022). How Templated Requirements Specifications Inhibit Creativity in Software Engineering. IEEE Transactions on Software Engineering, 48(10), 4074–4086. https://doi.org/10.1109/TSE.2021.3112503

Ó Cinnéide, M., Yamashita, A., & Counsell, S. (2016). Measuring refactoring benefits: A survey of the evidence. Proceedings of the 1st International Workshop on Software Refactoring, 9–12. https://doi.org/10.1145/2975945.2975948

Pascarella, L., Spadini, D., Palomba, F., & Bacchelli, A. (2019). On The Effect Of Code Review On Code Smells. https://doi.org/10.48550/arXiv.1912.10098

Pecorelli, F., Palomba, F., Khomh, F., & De Lucia, A. (2020). Developer-Driven Code Smell Prioritization. Proceedings of the 17th International Conference on Mining Software Repositories, 220–231. https://doi.org/10.1145/3379597.3387457

Peruma, A., AlOmar, E. A., Newman, C. D., Mkaouer, M. W., & Ouni, A. (2022). Refactoring Debt: Myth or Reality? An Exploratory Study on the Relationship Between Technical Debt and Refactoring (arXiv:2203.05660). arXiv. http://arxiv.org/abs/2203.05660

Sae-Lim, N., Hayashi, S., & Saeki, M. (2018). Context-based approach to prioritize code smells for prefactoring. Journal of Software: Evolution and Process, 30(6), e1886. https://doi.org/10.1002/smr.1886

Samarthyam, G., Suryanarayana, G., & Sharma, T. (2016). Refactoring for software architecture smells. Proceedings of the 1st International Workshop on Software Refactoring, 1–4. https://doi.org/10.1145/2975945.2975946

Smith, S. (2023, February 21). Architect modern web applications with ASP.NET Core and Azure. https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/

Sobhy, D., Bahsoon, R., Minku, L., & Kazman, R. (2021). Evaluation of Software Architectures under Uncertainty. ACM Transactions on Software Engineering and Methodology (TOSEM). https://doi.org/10.1145/3464305

Sumner, A., & Yuan, X. (2019). Mitigating Phishing Attacks: An Overview. Proceedings of the 2019 ACM Southeast Conference, 72–77. https://doi.org/10.1145/3299815.3314437

Suryanarayana, G., Samarthyam, G., & Sharma, T. (2014). Refactoring for Software Design Smells: Managing Technical Debt (1st edition). Morgan Kaufmann.

Sutter, T., Bozkir, A. S., Gehring, B., & Berlich, P. (2022). Avoiding the Hook: Influential Factors of Phishing Awareness Training on Click-Rates and a Data-Driven Approach to Predict Email Difficulty Perception. IEEE Access, 10, 100540–100565. https://doi.org/10.1109/ACCESS.2022.3207272

Traini, L., Di Pompeo, D., Tucci, M., Lin, B., Scalabrino, S., Bavota, G., Lanza, M., Oliveto, R., & Cortellessa, V. (2021). How Software Refactoring Impacts Execution Time. ACM Transactions on Software Engineering and Methodology (TOSEM). https://doi.org/10.1145/3485136

Vijayalakshmi, M., Mercy Shalinie, S., Yang, M. H., & U., R. M. (2020). Web phishing detection techniques: A survey on the state-of-the-art, taxonomy and future directions. IET Networks, 9(5), 235–246. https://doi.org/10.1049/iet-net.2020.0078

Wash, R. (2020). How Experts Detect Phishing Scam Emails. Proceedings of the ACM on Human-Computer Interaction, 4(CSCW2), 160:1-160:28. https://doi.org/10.1145/3415231

Wohlfarth, S., & Riebisch, M. (2006). Evaluating alternatives for architecture-oriented refactoring. 13th Annual IEEE International Symposium and Workshop on Engineering of Computer-Based Systems (ECBS'06), 7 pp. – 79. https://doi.org/10.1109/ECBS.2006.40

Zhuo, S., Biddle, R., Koh, Y. S., Lottridge, D., & Russello, G. (2022). SoK: Human-Centered Phishing Susceptibility. ACM Transactions on Privacy and Security. https://doi.org/10.1145/3575797

Evans, E. (2003). Domain-Driven Design: Tackling Complexity in the Heart of Software (1st edition). Addison-Wesley Professional.

Fowler, M. (2002). Patterns of Enterprise Application Architecture (1st edition). Addison-Wesley Professional.

Martin, R. C. (2008). Clean Code: A Handbook of Agile Software Craftsmanship (1st edition). Pearson.