**Universidad Tecnológica de La Habana**

**“José Antonio Echeverría”**



**Facultad de Ingeniería Informática**

**Migración de TestSoft a un modelo de software como servicio**

Migración de la interfaz gráfica para la gestión de test psicológicos y secciones de administración

*Trabajo para optar por el título de Ingeniería Informática*

**Autores:** Ernesto Alejandro Carralero Conde

Jesús Manuel Castellanos Reynaldo

**Tutores:** Ing. Tony García Álvarez

Dr. C. Margarita André Ampuero

**La Habana, Cuba**

**Junio, 2025**

**Resumen**

Los recursos humanos son reconocidos por los expertos como uno de los componentes más valiosos de las organizaciones en la actualidad. Los test psicológicos son ampliamente utilizados como apoyo en los procesos de gestión del capital humano. TestSoft es un software para la creación, aplicación, procesamiento, visualización y análisis de resultados de test psicológicos, que surge para ofrecer una solución a las limitantes de las aplicaciones ya existentes relacionadas con la temática. Además de presentar ciertas dificultades de usabilidad, potenciales clientes de la herramienta no poseen la infraestructura necesaria para el uso del software, por lo que, en base a este problema, surge el objetivo del presente trabajo: realizar una migración a un modelo de Software como Servicio (SaaS) y aprovechar la misma para solucionar las cuestiones antes mencionadas. Se propuso un diseño físico de base de datos para que TestSoft cumpliese con los requisitos del modelo SaaS, se realizó la migración de la interfaz gráfica de la aplicación a Vue.js y la API a NestJs. Se validó la solución a través de la ejecución de los test implementados y la comparación de los resultados brindados con respecto a la anterior interfaz, así como se probó la creación de test psicológicos y su funcionamiento al ser ejecutados. Como resultado se obtuvo una versión de TestSoft con mejor usabilidad y un diseño efectivo para el funcionamiento del modelo SaaS en la aplicación.

**Palabras claves:** Software como servicio, TestSoft, procesamiento y visualización de resultados de test psicológicos, procesos de gestión del capital humano

**Abstract**

Human resources are recognized by experts as one of the most valuable components of organizations today. Psychological tests are widely used to support human capital management processes. TestSoft is a software for creating, administering, processing, visualizing, and analyzing the results of psychological tests, which emerges to offer a solution to the limitations of existing applications related to the topic. In addition to presenting certain usability challenges, potential clients of the tool lack the necessary infrastructure for software usage. Therefore, based on this problem, the objective of this work emerges: to migrate to a Software as a Service (SaaS) model and leverage it to address the aforementioned issues. A physical database design was proposed to ensure TestSoft meets the requirements of the SaaS model, and the migration of the application's graphical interface began, specifically the execution of Belbin and MBTI tests and visualization of their results, as well as the process of creating psychological tests. The solution was validated through the execution of the implemented tests and comparison of the results provided in relation to the previous interface, as well as the creation of psychological tests and their functioning when executed. As a result, a version of TestSoft was obtained with better usability and an effective design for the functioning of the SaaS model in the application.

**Keywords**: Software as a service, TestSoft, processing and visualization of psychological test results, human capital management processes.

**Índice**

[Introducción 1](#_Toc200631970)

[Capítulo 1: Fundamentación teórica 8](#_Toc200631971)

[1.1 Introducción 8](#_Toc200631972)

[1.2 Caracterización de la herramienta Testsoft 8](#_Toc200631973)

[1.4 Deficiencias de la herramienta TestSoft 9](#_Toc200631974)

[1.5 Modelo Cloud Computing 12](#_Toc200631975)

[1.6 Software como servicio 13](#_Toc200631976)

[1.7 Arquitectura Multi-tenant 15](#_Toc200631977)

[1.8 Análisis de trabajos relacionados 16](#_Toc200631978)

[1.9 Justificación de la migración 19](#_Toc200631979)

[1.10 *Framework de front-end* 19](#_Toc200631980)

[1.10 *Framework* de *back-end* 20](#_Toc200631981)

[1.11 Conclusiones parciales 22](#_Toc200631982)

[Capítulo 2: Propuesta de solución 24](#_Toc200631983)

[2.1 Introducción 24](#_Toc200631984)

[2.2 Requisitos funcionales 24](#_Toc200631985)

[2.3 Requisitos de calidad 26](#_Toc200631986)

[2.4 Diagrama de casos de uso del sistema 27](#_Toc200631987)

[2.5 Arquitectura de la solución propuesta 29](#_Toc200631988)

[2.6 Patrones de diseño utilizados 33](#_Toc200631989)

[2.6.1 *Strategy* 34](#_Toc200631990)

[2.6.2 *Builder* 34](#_Toc200631991)

[2.6.3 *Facade* 35](#_Toc200631992)

[2.7 Propuesta de rediseño de base de datos 36](#_Toc200631993)

[2.8 Migración de la interfaz gráfica 42](#_Toc200631994)

[2.8.1 Arquitectura del proyecto de *front-end* 43](#_Toc200631995)

[2.8.2 Consideraciones en el diseño de la interfaz 45](#_Toc200631996)

[2.9 Migración de la API 49](#_Toc200631997)

[2.9.1 Arquitectura del proyecto de *back-end* 50](#_Toc200631998)

[2.10 Despliegue de la solución 52](#_Toc200631999)

[2.11 Conclusiones parciales 52](#_Toc200632000)

[Capítulo 3: Validación de la solución 54](#_Toc200632001)

[3.1 Introducción 54](#_Toc200632002)

[3.2 Casos de prueba funcionales 54](#_Toc200632003)

[3.2.1 Caso de Prueba: Recibo de los test asignados a un usuario 54](#_Toc200632004)

[3.2.2 Caso de Prueba: Ejecutar Test de Belbin 56](#_Toc200632005)

[3.2.3 Caso de Prueba: Ejecutar Test MBTI 57](#_Toc200632006)

[3.2.5 Caso de Prueba: Creación de copia del test Belbin 60](#_Toc200632007)

[3.3 Pruebas de rendimiento 61](#_Toc200632008)

[3.4 Conclusiones parciales 63](#_Toc200632009)

[Conclusiones 64](#_Toc200632010)

[Recomendaciones 65](#_Toc200632011)

[Referencias bibliográficas 66](#_Toc200632012)

**Índice de figuras**

[Figura 1: Fragmento de interfaz de la consulta de resultados en la herramienta. 10](#_Toc200632013)

[Figura 2: Fragmento de la interfaz de ejecución del test de Belbin en la herramienta. 11](#_Toc200632014)

[Figura 3: Fragmento de la interfaz de la ejecución del test MBTI en un dispositivo móvil. 12](#_Toc200632015)

[Figura 4: Diagrama de casos de uso del sistema 28](#_Toc200632016)

[Figura 5: Representación del patrón N-Capas en el proyecto. 32](#_Toc200632017)

[Figura 6: Representación del patrón *Strategy* en el proyecto. 34](#_Toc200632018)

[Figura 7: Representación del patrón *Builder* en el proyecto. 35](#_Toc200632019)

[Figura 8: Representación del patrón *Facade* en el proyecto. 36](#_Toc200632020)

[Figura 9**:** Fragmento del modelo físico de base de datos donde se muestran los cambios en las asignaciones de usuarios en grupos 38](#_Toc200632021)

[Figura 10: Fragmento del modelo físico de la base de datos. En rojo, los nuevos campos añadidos en relación al grupo donde se hizo la selección. 39](#_Toc200632022)

[Figura 11: Fragmento del modelo físico de la base de datos donde se muestran los nuevos atributos de la tabla *test\_application*. 40](#_Toc200632023)

[Figura 12: Fragmento del modelo físico de la base de datos donde se muestran las nuevas relaciones entre test, grupos y usuarios. 41](#_Toc200632024)

[Figura 13: Jerarquía de directorios en el proyecto *front-end* 43](#_Toc200632025)

[Figura 15: Fragmento de la interfaz de ejecución del test de Belbin en la nueva versión del software. 45](#_Toc200632026)

[Figura 16: Fragmento de la interfaz de ejecución del test de Belbin (muestra los puntos restantes de la pregunta al asignar puntos a una de sus respuestas). 46](#_Toc200632027)

[Figura 17: Fragmento de interfaz de ejecución del test MBTI (muestra mensajes indicando preguntas sin responder). 47](#_Toc200632028)

[Figura 18: Fragmento de interfaz del test MBTI (se señala en rojo la pregunta sin responder) 47](#_Toc200632029)

[Figura 19: Proceso de creación de series y preguntas 48](#_Toc200632030)

[Figura 20: Fragmento de la interfaz para la creación de un test 48](#_Toc200632031)

[Figura 21: Fragmento de la interfaz para la creación de un test (Muestra las condiciones en tiempo real, que deben cumplirse para avanzar al siguiente paso de creación del test). 49](#_Toc200632032)

[Figura 22: Jerarquía de directorios del proyecto *back-end* 50](#_Toc200632033)

[Figura 23: Diagrama de despliegue 52](#_Toc200632034)

[Figura 24: Fragmento de interfaz de selección de test a ejecutar. 55](#_Toc200632035)

[Figura 25: a) Fragmento de la interfaz de visualización de resultados del test de Belbin en la interfaz antigua (sección superior). b) Fragmento de la interfaz de visualización de resultados del test de Belbin en la nueva interfaz (sección inferior). 57](#_Toc200632036)

[Figura 26: a) Fragmento de la interfaz de visualización de resultados del test de MBTI en la interfaz antigua (sección izquierda). b) Fragmento de la interfaz de visualización de resultados del test de MBTI en la nueva interfaz (sección derecha). 59](#_Toc200632037)

[Figura 27: a) Fragmento de la interfaz de visualización de resultados de test en la interfaz antigua (sección izquierda). b) Fragmento de la interfaz de visualización de resultados del test en la nueva interfaz (sección derecha). 60](#_Toc200632038)

**Índice de tablas**

[Tabla 1: Comparativa de arquitecturas *Multi-tenant* 16](#_Toc200632039)

[Tabla 2: Comparación de trabajos relacionados 18](#_Toc200632040)

[Tabla 3: Caso de Prueba: Recibo de los test asignados a un usuario 54](#_Toc200632041)

[Tabla 4: Caso de Prueba: Ejecutar Test de Belbin 56](#_Toc200632042)

[Tabla 5: Caso de Prueba: Ejecutar Test MBTI 57](#_Toc200632043)

[Tabla 6: Caso de Prueba: Consulta de resultados de test MBTI 59](#_Toc200632044)

[Tabla 7: Caso de Prueba: Creación de copia del test Belbin 60](#_Toc200632045)

[Tabla 8: Resultados de pruebas de rendimiento realizadas a ambas versiones de la aplicación 62](#_Toc200632046)

# Introducción

En el mundo actual, los recursos humanos desempeñan un papel fundamental en el éxito y la sostenibilidad de las organizaciones. Las empresas evolucionan y se enfrentan a un entorno empresarial cada vez más competitivo, por tanto, el reclutamiento de personal competente y efectivo se torna más importante para lograr resultados positivos [1-3].

Los test psicológicos desempeñan un papel crucial en la gestión de recursos humanos. Los test proporcionan información valiosa sobre las capacidades, habilidades, rasgos de personalidad y competencias de los individuos. Ellos ofrecen datos objetivos para tomar decisiones informadas sobre la idoneidad de un candidato para un puesto específico. Además, se pueden utilizar para evaluar el desempeño de un empleado y determinar habilidades claves que pueden influir en su rol dentro de la empresa [4, 5].

Las evaluaciones psicológicas ayudan a construir equipos equilibrados lo que posibilita un aumento del rendimiento del equipo y contribuye a mantener un ambiente de trabajo saludable y armonioso [4].

La automatización del proceso de aplicación de tests psicológicos es una tendencia cada vez más relevante en el ámbito de la psicología y la tecnología educativa, y ofrece numerosas ventajas. Uno de los beneficios más destacados es la reducción del consumo de materiales físicos, como el papel, ya que los tests digitales eliminan la necesidad de imprimir cuestionarios, hojas de respuesta y materiales de apoyo. Esto no solo contribuye a la sostenibilidad ambiental, sino que también facilita el almacenamiento y organización de las respuestas en bases de datos, evitando el uso de archivos físicos y reduciendo el riesgo de pérdida o deterioro de la información [6].

La automatización permite un ahorro significativo de tiempo, tanto para los profesionales como para los participantes. Por un lado, los psicólogos pueden dedicar menos tiempo a la administración manual de los tests y enfocarse en el análisis e interpretación de los resultados. Por otro lado, los participantes pueden completar los tests en línea en cualquier momento y lugar, lo que elimina la necesidad de desplazamientos y coordinación de horarios. Esta flexibilidad no solo mejora la experiencia del usuario, sino que también amplía el alcance de los tests, permitiendo que personas en áreas remotas o con movilidad reducida puedan acceder a ellos [6, 7].

Otro aspecto importante es la estandarización y precisión que ofrecen los sistemas automatizados. Al garantizar que los tests se apliquen de manera uniforme, se reducen los errores humanos en la administración y calificación. Además, la calificación automática minimiza los sesgos y errores en la interpretación de las respuestas, lo que aumenta la confiabilidad de los resultados. Esto es especialmente relevante en contextos donde se requiere una evaluación rigurosa y objetiva, como en la selección de personal o en la investigación psicológica [6].

Los tests en línea pueden llegar a una población más amplia, incluyendo personas que, por diversas razones, no podrían participar en una evaluación presencial. Asimismo, la automatización permite administrar tests a grandes grupos de manera simultánea, lo que es especialmente útil en contextos educativos o laborales donde se requiere evaluar a un número elevado de personas en un corto período de tiempo [7].

Los tests psicológicos son herramientas valiosas en la práctica profesional de los psicólogos, pero es importante destacar que su función es de apoyo y no deben considerarse como una guía definitiva. Estos instrumentos proporcionan datos cuantificables y estandarizados que pueden ayudar a identificar tendencias, rasgos o patrones de comportamiento en los individuos. Sin embargo, su interpretación debe ser complementada con el juicio clínico y la experiencia del profesional, ya que los resultados de los tests no capturan la complejidad única de cada persona. Factores como el contexto cultural, el estado emocional del individuo en el momento de la evaluación y las circunstancias específicas de su vida pueden influir en los resultados, lo que requiere un análisis más profundo y personalizado. Por lo tanto, los tests psicológicos son una pieza más dentro de un proceso de evaluación integral, y su uso debe ir acompañado de entrevistas, observaciones y otras técnicas que permitan al psicólogo formarse una visión completa y precisa del individuo [8].

Existen diversas herramientas de software diseñadas para la ejecución de uno o varios tests psicológicos y de aptitud. Estas herramientas se dividen en dos categorías principales: aquellas basadas en la web y las que son aplicaciones de escritorio. Entre las herramientas web, se encuentran plataformas como *Psychometric Success* [9] y *Open Source Psychometrics Project* [10], mientras que en el ámbito de escritorio destaca el *Psychology Experiment Building Language* [11].

En cuanto a las herramientas web, algunas requieren el pago de una suscripción para acceder a la ejecución de tests completos. Por ejemplo, sitios como *HRPersonality*[12] y *e-TEAediciones*[13] ofrecen sus servicios bajo un modelo de pago. Por otro lado, hay plataformas que permiten realizar algunos tests de manera gratuita como prueba, pero para acceder a la totalidad de los tests, es necesario realizar un pago. Un caso representativo de esto es *Practice Aptitude Tests*[14]. Por otro lado, existen herramientas tanto web como de escritorio que ofrecen la ejecución de tests de forma gratuita. Sin embargo, plataformas, como *MBTIonline*[15] solo permiten la ejecución de un único test, en este caso, el Myers-Briggs Type Indicator (MBTI).

A pesar de la disponibilidad de estas herramientas, existe una limitación significativa: la mayoría de ellas solo brindan los resultados al usuario en el momento de la ejecución y no permiten la recuperación posterior de estos resultados. Esto significa que no es posible acceder a los resultados en una fecha posterior, ni para un usuario individual ni para un grupo de usuarios. Además, ninguna de estas herramientas ofrece funcionalidades avanzadas que permitan analizar los resultados de manera conjunta para un grupo seleccionado. Aquellas plataformas que sí ofrecen esta capacidad de análisis grupal, como *Practice Aptitude Tests* [14], lo hacen bajo un modelo de pago.

En resumen, si bien existen diversas herramientas disponibles para la ejecución de tests psicológicos y de aptitud, la mayoría de ellas presentan limitaciones en cuanto a la recuperación de resultados y la posibilidad de análisis grupal. Aquellas que sí ofrecen estas funcionalidades avanzadas suelen requerir una suscripción o pago, lo que limita su accesibilidad para aquellos que buscan herramientas gratuitas y completas.

Para compensar estas dificultades, surge **TestSoft**, que es un software para la creación, aplicación, procesamiento y visualización de test psicológicos y psicométricos a grupos de personas en una institución. Desarrollada en el año 2018 en la Facultad de Ingeniería Informática de la CUJAE, TestSoft garantiza que se incorporen nuevos test sin necesidad de modificar el código de la aplicación, pues ofrece funcionalidades para configurarlos según parámetros establecidos [16].

En el 2021 se implementa una nueva versión de TestSoft que posibilita la incorporación de nuevos tipos de test y mejora el procesamiento de los resultados, así como determinadas mejoras en el aspecto de usabilidad de la interfaz de usuario y la validación de los datos de entrada [17].

No obstante, en vista a un uso comercial, la herramienta cuenta con determinadas deficiencias que dificultan su despliegue según las necesidades de potenciales clientes, entre ellas se encuentran [18]:

* Actualmente es necesario que el cliente posea una infraestructura propia donde poder desplegar y utilizar la herramienta.
* En correspondencia con el punto anterior, actualizar la aplicación implica actualizar el software a cada cliente por separado, operación que resulta tediosa y costosa.
* El proyecto requiere que los datos recopilados por el uso de la aplicación por parte de los clientes, sean visibles al proveedor de la herramienta, con el objetivo de realizar investigaciones y análisis posteriores.
* La interfaz gráfica posee aún dificultades de usabilidad, y algunos de los componentes de validación de respuestas de los test, son obsoletos o no poseen un funcionamiento adecuado en todos los entornos.

Como resultado del análisis de estos aspectos, se identifica el siguiente **problema a resolver:** ¿Cómo no involucrar a los clientes de TestSoft en el despliegue de la herramienta dada la no disponibilidad de infraestructura por parte de los mismos?

Para resolver este problema se definió como **objeto de estudio** las herramientas informáticas para la ejecución de test psicológicos, el área del *Cloud Computing* y el modelo SaaS y como **campo de acción,** el desarrollo de la herramienta TestSoft bajo el modelo de software como servicio.

Como **objetivo general** se ha decidido migrar TestSoft a un modelo de software como servicio (SaaS).

El objetivo general requiere del cumplimiento de los siguientes **objetivos específicos**:

1. Identificar las insuficiencias de TestSoft que limitan utilizar la herramienta como servicio.
2. Rediseñar la base de datos para que permita brindar la herramienta como servicio.
3. Migrar la interfaz gráfica y con ello solucionar los problemas de usabilidad de la aplicación.
4. Desarrollar la API para lograr una conexión satisfactoria entre el cliente y las funcionalidades.
5. Validar la propuesta a través del desarrollo de casos de prueba para el testeo y posterior corrección de errores que puedan surgir.

Para dar cumplimiento a los objetivos específicos se establece las siguientes **tareas**:

1. Asimilar la herramienta TestSoft (Jesús y Ernesto).
2. Asimilar la base de datos de herramienta TestSoft. (Jesús y Ernesto).
3. Asimilar *framework* y lenguajes de desarrollo web de la aplicación (Jesús y Ernesto).
4. Asimilar modelo de distribución de software como servicio (Jesús y Ernesto).
5. Realizar ajustes al diseño físico de la base de datos de manera que soporte el funcionamiento de la herramienta como una solución SaaS (Jesús y Ernesto).
6. Documentar la base de datos de TestSoft, precisando la tablas y campos que fueron modificados (Jesús y Ernesto).
7. Documentar los servicios web que actualmente están implementados en la herramienta (Jesús y Ernesto).
8. Implementar la visualización de los test disponibles para un usuario a partir de su autenticación en la aplicación (Jesús).
9. Crear los componentes necesarios para la ejecución de los test (Jesús)
10. Implementar la funcionalidad para guardar la ejecución de los test, al recibir los datos a través de la API (Ernesto).
11. Implementar la vista de muestreo de los resultados de los test realizados por el usuario (Jesús).
12. Implementar la gestión de test psicológicos (Jesús y Ernesto).
13. Implementar la gestión de usuarios (Jesús y Ernesto).
14. Implementar la gestión de clientes (Jesús y Ernesto).
15. Implementar la gestión de grupos (Jesús y Ernesto).
16. Implementar la gestión de accesos y permisos (Jesús y Ernesto).
17. Gestionar la configuración de versiones de las tareas implementadas a través de la herramienta GitHub (Jesús y Ernesto).
18. Diseñar los casos de prueba que permitan validar la ejecución y muestreo de los test implementados (Jesús y Ernesto).
19. Ejecutar los casos de prueba y solucionar errores para validar la solución implementada (Jesús y Ernesto).

El informe se encuentra estructurado en introducción, tres capítulos, conclusiones, recomendaciones y referencias bibliográficas.

En el Capítulo 1 se realiza un análisis de la herramienta TestSoft, identificando las insuficiencias, se ofrecen definiciones del modelo SaaS y la arquitectura multi-tenant, así como sus características, ventajas y desventajas. Por último, se fundamentan las tecnologías a utilizar para el desarrollo de la solución.

En el Capítulo 2 se documenta la propuesta de diseño de base de datos para que la herramienta cumpla con los requisitos SaaS, la estructura del proyecto de *front-end*, patrones de diseño y arquitectónicos usados, las vistas y componentes implementados con respecto a la creación y ejecución de los test, y se explican las mejoras de usabilidad en la interfaz de la herramienta con respecto a la anterior interfaz.

El Capítulo 3 propone la validación de la solución propuesta a través de casos de prueba como la comprobación de la igualdad de los resultados entregados por parte de la nueva aplicación con respecto a la anterior, el correcto funcionamiento de la validación de los componentes implementados, y la revisión del comportamiento de la aplicación en casos extremos como el agotamiento del tiempo destinado para la ejecución de los test. Además, se hicieron pruebas de rendimiento para comprobar la velocidad de la nueva versión del proyecto *back-end*.

# Capítulo 1: Fundamentación teórica

## 1.1 Introducción

En este capítulo se realiza un análisis detallado de la herramienta TestSoft, propósito y evolución, así como las tecnologías empleadas en su desarrollo. Además, se definen los conceptos fundamentales del modelo SaaS y la arquitectura *multi-tenant*, junto con un análisis de sus características, beneficios y desafíos inherentes. Se exploran los conceptos de *frameworks* de *front-end* y de *back-end* y se ofrece una revisión de trabajos relacionados. Por último, se fundamentan las tecnologías a utilizar para dar solución a la problemática descrita.

## 1.2 Caracterización de la herramienta Testsoft

Las organizaciones manejan diversos recursos, entre los más influyentes en su correcto funcionamiento se encuentra los recursos humanos (RH). Se entiende que los RH deben ser debidamente gestionados ya que se identifican como sumamente valiosos [1-5, 16].

Las organizaciones deben contar con un procedimiento documentado para el proceso de selección de sus recursos humanos, para ello aplican diferentes métodos, técnicas y herramientas. Comúnmente, se utilizan experiencias anteriores, certificaciones de notas y títulos universitarios. Uno de los métodos más utilizados por las organizaciones para captar, asignar y desarrollar correctamente sus recursos son los test psicológicos, los cuales se emplean para medir el nivel de capacitación individual y la interacción entre los individuos [4, 5, 16].

La aplicación manual de los tests se convierte en un proceso complicado, tedioso y lento, muy propenso a errores. Las herramientas existentes para la automatización de este proceso, presentaban algunas dificultades como no permitir realizar análisis de los resultados de los test por grupos de usuarios [16].

A partir de esta situación se concluyó que era necesario desarrollar una herramienta que permita configurar la creación, procesamiento y visualización de diferentes test, su aplicación y el análisis de los resultados de manera que sirva de apoyo a los procesos de gestión del capital humano [16].

De esta manera surge en 2018 la herramienta TestSoft, posteriormente actualizada y mejorada en 2021 [16, 17].

TestSoft es una aplicación web desarrollada con el lenguaje PHP en el *framework* Yii2. Esto favorece la centralización de los datos y el acceso a la herramienta desde cualquier dispositivo en cualquier momento [16].

La herramienta sigue una arquitectura monolítica de modelo vista controlador (MVC). Todo el código de la interfaz visual y lógica se encuentra junto en el mismo proyecto, aunque separado en distintas capas estrechamente relacionadas [16]:

* Modelo: encapsula los datos y las funcionalidades del sistema, y es independiente de cualquier representación de salida y comportamiento de entrada.
* Vista: muestra la información al usuario a través de las pantallas de la aplicación. Pueden existir varias vistas y cada una tiene asociada un componente controlador.
* Controlador: es el encargado de capturar las solicitudes HTTP y enviarlas a los *actions*, quienes las traducen a solicitudes para el modelo o la vista.

Esto implica que el *back-end* de la aplicación sea *statefull,* o sea, que el acceso de los usuarios a la aplicación y su autorización a sus distintas funcionalidades son controladas mediante sesiones que son guardadas del lado del servidor [18].

La base de datos encargada de almacenar la información pertinente está implementada con MySQL**,** el cual es un sistema de gestión de bases de datos relacional [16-18].

## 1.4 Deficiencias de la herramienta TestSoft

El proveedor afirma que la herramienta presenta errores al ejecutarse en otro tipo de navegadores, problema de gran importancia para el objetivo de la aplicación, pues se requiere que pueda ser ejecutada en diversos entornos para poder ser utilizada por mayor cantidad de clientes [18].

La herramienta en su estado actual, presenta secciones en las que su usabilidad es mejorable, según la experiencia de uso. Existen ciertos puntos que deben ser perfeccionados para posibilitar una experiencia de usuario confortable y lo suficientemente intuitiva como para que el cliente pueda comprender las funcionalidades de la aplicación:

* Durante la ejecución de un test, para poder visualizar el tiempo restante, es necesario regresar a la sección superior de la página, hecho que resta tiempo y comodidad al usuario.
* El apartado de navegación entre series ocupa demasiado espacio en la pantalla, y ocurre lo mismo con al temporizador, por lo tanto, sería ideal poder navegar entre series en cualquier momento.



Figura 1: Fragmento de interfaz de la consulta de resultados en la herramienta.

* La forma de mostrar los resultados de los test, a pesar de ser objetiva y concisa, puede ser mejorada para que el usuario pueda comprenderla con más facilidad. Además, para acceder a los resultados personales de un usuario, hay componentes que podrían ser eliminados para una obtención más directa de los mismos, y se podría optimizar el uso de los espacios disponibles en pantalla para visualizarlos. Esto se puede apreciar en la Figura 1, donde la interfaz obliga al usuario a seleccionarse a sí mismo en una tabla de una fila, y luego seleccionar un test, para así, finalmente, poder consultar sus resultados.
* El test de Belbin presenta un componente para poder distribuir 10 puntos entre varias respuestas de valores numéricos. Actualmente se está utilizando un componente de introducción de texto, pero sería factible brindar además la posibilidad de incrementar o disminuir los valores a través de botones. Además, al introducir un valor superior al posible en una respuesta, la aplicación informa del error y regresa el valor al estado anterior, por lo cual se propone como solución para el ahorro del tiempo del test la asignación directa del máximo valor posible en cuestión. Como se puede apreciar en la Figura 2, los inputs a la derecha no poseen botones para incremento o decremento, y no se intuye correctamente que el número 10, en la parte superior derecha, sea la cantidad de puntos restantes en la pregunta.

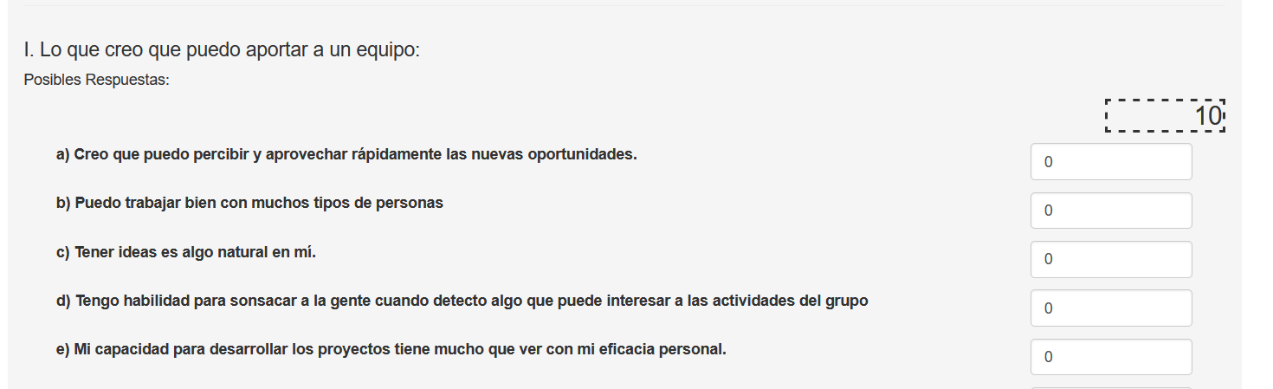


Figura 2: Fragmento de la interfaz de ejecución del test de Belbin en la herramienta.

* Las preguntas y respuestas en el test MBTI se encuentran muy unidas entre sí, lo que puede afectar la diferenciación entre las mismas y causar confusión en el usuario. Además, la versión *responsive* de esta vista no se encuentra implementada, lo que constituye un problema para los clientes que quieran ejecutar el software en dispositivos de pantalla estrecha, como se muestra en la Figura 3.

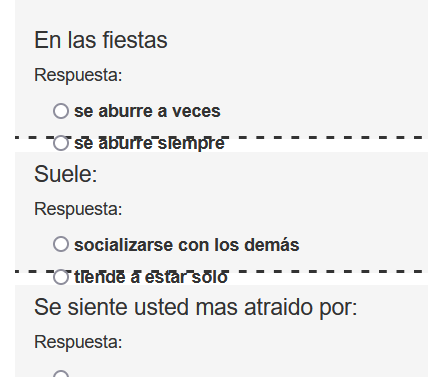


Figura 3: Fragmento de la interfaz de la ejecución del test MBTI en un dispositivo móvil.

* El proceso de creación de un test resulta poco intuitivo y difícil de comprender para el usuario promedio. Además, existen ciertas validaciones que no funcionan como deberían.

En las condiciones actuales de la herramienta, más específicamente de la base de datos, es necesario que la herramienta sea desplegada por especialistas en una infraestructura proporcionada por los clientes. Consecuentemente, al ocurrir una actualización, habría que realizar un nuevo despliegue por cada uno de los clientes que desearan la actualización [18].

Esto requiere, como se menciona anteriormente, que los clientes posean una infraestructura tecnológica interna propia para poder realizar el despliegue de la herramienta en su institución. Esta infraestructura se compone de equipo de hardware costoso, lo que, en caso de no poseerlo, implica una inversión extra además de la adquisición del software.

Por estas razones, se decide la implementación de un modelo de software como servicio, el cual constituye una rama del *Cloud Computing,* tema tratado a continuación.

## 1.5 Modelo Cloud Computing

El *Cloud Computing*, o computación en la nube, es un modelo que ofrece acceso a recursos informáticos compartidos a través de la web, como servidores, almacenamiento y software. En lugar de mantener la infraestructura localmente, los clientes pueden acceder a estos servicios a través de proveedores de servicios en línea [19].

Los usuarios pueden aprovisionar y gestionar recursos de forma automática a través de una interfaz en línea sin necesidad de intervención humana del proveedor. Los servicios están disponibles a través de la red y son accesibles desde cualquier lugar con conexión a Internet [20, 21].

Los recursos informáticos del proveedor en este tipo de servicios, por lo general se comparten entre múltiples clientes, elemento que permite la optimización y la eficiencia en el uso de los mismos. Los servicios *online* pueden escalar automáticamente para adaptarse a las necesidades de capacidad, lo que posibilita aumentar o disminuir recursos según la demanda del cliente, ya que ellos pagan por el uso de recursos en función de la cantidad consumida. Esto proporciona flexibilidad y control de costos [21].

Algunos de los servicios en la nube más utilizados son [19, 20]: Amazon Web Services (AWS), Microsoft Azure, Google Cloud Plataform, Salesforce y Dropbox.

## 1.6 Software como servicio

El Software como servicio (SaaS, por sus siglas en inglés) es un modelo de distribución de software que consiste en proveer a través de Internet, una aplicación como un servicio a diversos clientes [19].

Al remover la necesidad de instalar la aplicación, brinda a los proveedores una rentabilidad mayor que la de los softwares convencionales. Las aplicaciones de arquitectura SaaS se encuentran en auge en la actualidad, y poseen tendencia a un mayor uso por las facilidades que ofrecen. La mayoría de los proveedores de este servicio ofrecen a sus clientes planes de pago mensuales [22].

Los SaaS han transformado la forma en que las empresas y los usuarios consumen y acceden a las aplicaciones [23].

**Ventajas y desventajas del modelo SaaS**

El uso del modelo SaaS tiene ventajas y desventajas con respecto a otras arquitecturas. Entre las ventajas más significativas se encuentran [19]:

* Permite acceder a las aplicaciones desde cualquier dispositivo con conexión. Esto permite que los clientes puedan usarlas desde cualquier lugar, e incluso prescindir de medios de cómputo para poder almacenarla.
* Ofrece escalabilidad flexible lo que posibilita a los clientes ajustar fácilmente el uso y los costos según sus necesidades, las cuales pueden variar en el transcurso del tiempo.
* Ofrece siempre la versión más actualizada del software, con lo que libera a los clientes de las tediosas y costosas tareas de actualización y mantenimiento.
* Posibilita la implementación rápida y sencilla de nuevas funcionalidades, característica que agiliza la innovación y atrae a los clientes a la adquisición del software.
* Promueve la colaboración en tiempo real entre equipos distribuidos geográficamente.

**Características de una aplicación SaaS**

Una aplicación para ser SaaS debe cumplir con varios requisitos [19]:

* Escalabilidad: La aplicación debe priorizar el funcionamiento eficiente, para reducir tiempos de carga y consumo de recursos del ordenador del cliente.
* Eficiencia *Multi-tenant*: Capacidad para compartir entre clientes la mayor cantidad de recursos posibles, pero a la vez de diferenciar entre los correspondientes a cada cual.
* Personalización: Ofrecer al cliente la posibilidad de personalizar su aplicación, sin incluir costos mayores de implementación.
* Aseguramiento de calidad: El proveedor de SaaS debe ofrecer un servicio de soporte adecuado a sus clientes, así como ser responsable de preservar la seguridad e integridad de los datos almacenados por ellos.

Como toda solución, a su vez posee desventajas o debilidades, entre las que se encuentran [19, 22, 24]:

* Los proveedores de SaaS tienen la necesidad de invertir en medidas de seguridad avanzadas para proteger los datos de los usuarios y cumplir con los estándares necesarios en términos de seguridad y privacidad.
* Requiere de una conexión a Internet constante, estable, y, en muchas ocasiones, poderosa para poder funcionar.
* Debe manejar la concurrencia de un número elevado de usuarios.
* Estar expuesto en línea lo convierte en objeto de ataques cibernéticos y de robos de información.
* La migración de sistemas web a arquitecturas de este tipo implica cambios fundamentales en la estructura del software en cuestión.

## 1.7 Arquitectura Multi-tenant

La arquitectura *Multi-tenant* consiste en albergar una instancia de software, a la cual varios ***tenants*** o clientes, accederán a través de la red. Entre sus principales características están la alta capacidad de personalización que ofrecen en dependencia de las necesidades de los clientes. A pesar de ser una única aplicación, los cambios realizados por un usuario no deben afectar a los demás ***tenants***[24].

Existen tres formas de implementar una arquitectura *Multi-tenant* que se basan en la forma en que se distribuye la base de datos del servicio [25]:

1. Bases de datos separadas: Esta alternativa ofrece una base de datos a cada cliente. Entre sus principales ventajas destaca la facilidad para diferenciar los datos de cada ***tenant***, y la personalización que brinda la posibilidad de poseer una base de datos particular. Como dificultad principal se encuentra el hecho de que el costo de soporte es mucho mayor.
2. Base de datos compartida, diferentes esquemas: Con esta opción, se utiliza una base de datos, pero se ofrece un esquema, o conjunto de tablas, distinto para cada cliente. Es relativamente fácil de implementar y soporta gran cantidad de clientes por servidor. Sin embargo, presenta la desventaja de que es más difícil restaurar la información en caso de pérdida por parte de un cliente, pues esto implicaría la sobre escritura completa de la base de datos, lo que afectaría la información de los otros ***tenants***.
3. Base de datos compartida, un esquema: Se utiliza una misma base de datos y esquema para todos los clientes. Esta alternativa ofrece el menor consumo de recursos de hardware y las mayores facilidades para realizar salvas de información, pero requiere esfuerzos adicionales por parte de los desarrolladores para evitar que los ***tenants*** puedan acceder a datos ajenos. El enfoque de esquema compartido es ideal cuando la aplicación debe servir a una cantidad considerable de clientes, con pocos servidores de parte del proveedor.

Al reunir las principales características de los diferentes tipos de arquitecturas *Multi-tenant* se puede obtener la Tabla 1.

Tabla 1: Comparativa de arquitecturas *Multi-tenant*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bases de Datos** | **Descripción** | **Diferenciación de datos** | **Consumo de Recursos** | **Salvas y restauración de información** |
| Separadas | Se utiliza una base de datos para cada cliente | Fácil | Alto | Fácil |
| Compartida (Diferentes Esquemas) | Se utiliza una base de datos, con esquemas diferentes para cada cliente | Fácil | Medio | Difícil |
| Compartida (Un Esquema) | Se utiliza una base de datos para todos los clientes | Difícil | Bajo | Fácil |

## 1.8 Análisis de trabajos relacionados

Como se observa en la Tabla 2, existen varias herramientas que sirven de soporte a la ejecución de diferentes test. La mayoría de las soluciones son web lo que posibilita la aplicación simultánea de test a un grupo de usuarios y su almacenamiento centralizado. Algunas herramientas exigen pagos para contar con los servicios y otras permiten ejecutar solo algunos test de forma gratuita. Otro conjunto es totalmente gratuito pero sus servicios son limitados. En su inmensa mayoría, con independencia de la cantidad de test que permitan ejecutar, solo brindan la posibilidad de obtener los resultados del test al usuario que lo ejecuta y en el momento de la ejecución. Esto limita la posibilidad de realizar análisis para diferentes grupos de usuarios tomando en cuenta diferentes criterios. Las herramientas que ofrecen servicios de análisis de los resultados de diferentes test y para grupos de usuarios seleccionados no son gratuitas. A pesar de que la gran mayoría de aplicaciones analizadas poseen un modelo de software como servicio, no poseen las cualidades de la versión actual de Testsoft.

Tabla 2: Comparación de trabajos relacionados

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Solución** | **Web/Escritorio** | **Permite añadir nuevos test** | **Procesamiento configurable** | **Devuelve resultado** | | **Realizar análisis estadísticos con diferentes criterios** | **SaaS** |
| **Individual** | **Contra todos los evaluados** |
| HRPersonality | **W** |  |  | **X** |  | **X** | **X** |
| e-TEAediciones | **W** |  |  | **X** |  | **X** | **X** |
| MBTIonline | **W** |  |  | **X** |  |  | **X** |
| Practice Aptitude Tests | **W** |  |  | **X** | **X** | **X** | **X** |
| Psychology Experiment Building Language (PEBL) | **E** | **X** | **X** | **X** |  |  |  |
| Google Forms | **W** | **X** |  |  |  |  | **X** |
| Testsoft | **W** | **X** | **X** | **X** | **X** | **X** |  |

Con el análisis anterior se puede concluir que es necesario llevar a un modelo SaaS a la herramienta para que además de ofrecer las ventajas que ya posee con respecto a los softwares planteados, pueda gozar de las cualidades que ofrece este modelo de negocio.

## 1.9 Justificación de la migración

Los clientes potenciales de la aplicación, como se ha descrito anteriormente, no poseen la infraestructura necesaria para utilizar la herramienta en su estado actual. A esto se añade la necesidad del proveedor de TestSoft de que los datos recopilados con el uso de la aplicación puedan ser analizados, para realizar investigaciones y estudios sobre el comportamiento de los recursos humanos. Se prevé que la aplicación sea desplegada en un servidor con ciertas limitaciones de espacio. Además, existen datos en la base de datos actual que deben ser conservados, por lo que se debe realizar la menor cantidad de modificaciones posibles. Por las razones anteriores, se decide proponer la alternativa de una aplicación SaaS *Multi-tenant*, con una base de datos compartida y un esquema.

La herramienta TestSoft se encuentra en un estado monolítico, lo cual presenta diversas desventajas para la migración a un modelo SaaS, como lo es, principalmente, la dificultad para su escalabilidad (aspecto de suma importancia para el modelo de software como servicio). En este sentido, el mantenimiento de los componentes, que por lo general se encuentran acoplados en una única estructura, se dificulta demasiado y por tanto se sugiere la separación de los mismos para que puedan escalar individualmente [26].

Por esta razón se decide migrar la aplicación a dos proyectos diferentes, uno *front-end* y otro *back-end*, y con ello se aprovecha la situación para mejorar y modernizar los componentes visuales y resolver las problemáticas de compatibilidad de navegadores.

## 1.10 *Framework de front-end*

Los *frameworks* de *front-end* son colecciones de código HTML, CSS y JavaScript estandarizado, escritas previamente, que proporcionan una estructura y funcionalidades predefinidas para facilitar el desarrollo de interfaces de usuario interactivas en aplicaciones web. Ellos permiten un desarrollo web más eficiente y estructurado, al optimizar el flujo de trabajo y la calidad del software. Suelen proporcionar un conjunto de herramientas y bibliotecas para crear interfaces de usuario, manejar eventos e interactuar con APIs. Ayudan a los desarrolladores a lograr coherencia entre proyectos, acelerar el desarrollo y reducir la complejidad del código [27] .

Entre los *frameworks* más populares y utilizados en la actualidad se encuentran: Vue.js, React.js y Angular.js. Estos *frameworks* se caracterizan por el uso de la reactividad, la cual facilita el desarrollo de aplicaciones web, ya que permiten que cualquier modificación en los datos se refleje automáticamente en la interfaz sin necesidad de intervención manual por parte del desarrollador. La presencia de esta característica es vital para la aplicación a desarrollar, donde se debe mostrar en tiempo real, respuestas del sistema al usuario [27-32].

Se decide seleccionar Vue.js para la implementación del nuevo *front-end* de la herramienta. Vuejs es un *framework* de JavaScript liviano, creado por Evan You. La popularidad de Vue.js se ha incrementado en los últimos años, dado por su baja curva de aprendizaje y alto rendimiento en aplicaciones relativamente sencillas, considerado superior al de las otras dos tecnologías mencionadas. Otro elemento a considerar en la decisión es el conocimiento previo de este *framework* por parte del equipo de trabajo [33].

## 1.10 *Framework* de *back-end*

Un *framework back-end* es un conjunto de herramientas, librerías y convenciones que facilitan el desarrollo de aplicaciones en el lado del servidor. Proporciona una estructura base para construir aplicaciones web, APIs, servicios o sistemas que manejen la lógica de negocio, la interacción con bases de datos, la autenticación, la seguridad y otras funcionalidades necesarias para el funcionamiento de una aplicación [34].

Ofrecen una arquitectura predefinida (como MVC) para organizar el código de manera eficiente, permiten reutilizar componentes comunes, como conexiones a bases de datos, autenticación y manejo de solicitudes HTTP, incluyen medidas para proteger la aplicación contra vulnerabilidades comunes, como inyecciones SQL o ataques XSS, y proporcionan herramientas para interactuar con bases de datos de manera sencilla [34, 35].

Entre los *frameworks* *back-end* más populares se encuentran Node.js, Express.js, NestJS, Django, FastAPI y Laravel [35].

Dada la alta concurrencia que presentan los sistemas SaaS, es necesario poner en consideración si Yii2 es la tecnología más adecuada para la implementación de un sistema orientado a este modelo.

*Node.js* es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación JavaScript, asíncrono, con entrada/salida de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, servidores web [36].

Como alternativa se propone *Node.js*, por las siguientes razones [36]:

1. Utiliza un modelo de programación asíncrono y no bloqueante, lo que le permite manejar múltiples solicitudes simultáneamente de manera eficiente.
2. La arquitectura basada en eventos de Node.js permite una mejor escalabilidad y rendimiento para aplicaciones en tiempo real.
3. Compila el código *JavaScript* a *bytecode*, lo que mejora significativamente el rendimiento.

No obstante, con Node.js por sí solo, el desarrollo puede hacerse lento y propenso a problemas de arquitectura y organización del proyecto [37].

*NestJs* es un *framework* de *NodeJs* progresivo para crear aplicaciones del lado del servidor eficientes, confiables y escalables, lo cual es fundamental para un modelo SaaS donde la carga puede variar significativamente según el número de usuarios y el uso de los servicios. El mismo utiliza el lenguaje TypeScript y se basa en conceptos como la inyección de dependencias y presenta una arquitectura modular, lo que puede facilitar el desarrollo y mantenimiento del código [37-39].

*NestJs*, al estar basado en *Node.js*, puede manejar un gran número de conexiones simultáneas gracias a su naturaleza asíncrona, lo que puede mejorar el rendimiento y la capacidad de respuesta de la aplicación [39].

Yii2 está basado en PHP, un lenguaje que no maneja las operaciones asíncronas de manera tan eficiente como *Node.js*. Esto puede resultar en un menor rendimiento en aplicaciones con alta concurrencia. El ecosistema de herramientas y bibliotecas de Yii2 es menos extenso comparado con el de *Node.js/NestJs*, lo que puede limitar las opciones y la flexibilidad en el desarrollo. Yii2 no está tan optimizado para arquitecturas de microservicios y escalabilidad horizontal como *NestJs*, lo cual puede ser un obstáculo en un entorno SaaS donde es crucial manejar un gran número de usuarios y datos. El código en Yii2 puede volverse complejo y difícil de mantener con el tiempo, especialmente en aplicaciones grandes, debido a la menor modularidad comparada con NestJs [40, 41].

Aunque Yii2 tiene una comunidad activa, la comunidad y los recursos disponibles para *Node.js* y *NestJs* son significativamente más grandes, lo que facilita la resolución de problemas y la adopción de buenas prácticas [40].

Por tanto, debido a las razones anteriores, se decide utilizar a *NestJs* como tecnología de desarrollo para el proyecto *back-end*.

## 1.11 Conclusiones parciales

Al concluir el capítulo se arriban a las siguientes conclusiones:

* Entre las herramientas para el desarrollo de test psicológicos destaca TestSoft, pues permite, por ejemplo, la creación de test de forma altamente configurable y análisis de los resultados obtenidos. Sin embargo, es preciso mejorar la usabilidad de la aplicación y lograr las condiciones necesarias para su uso por clientes que no poseen infraestructura interna propia.
* La computación en la nube y en especial, el modelo de negocio de software como servicio, reciben un auge en su uso por las ventajas que presentan: facilitan el acceso de los clientes a la aplicación, lo que posibilita un ahorro en recursos computacionales, así como un control más eficaz sobre el acceso a los servicios ofrecidos por los proveedores.
* Es preciso rediseñar la base de datos de TestSoft para lograr la separación de los datos de cada *tenant*, y con ello cumplir con los requisitos de un sistema SaaS *multi-tenant*.
* Para el rediseño de la base de datos se decide utilizar un modelo *multi-tenant* de Base de datos compartida con un solo esquema, por la necesidad de realizar la menor cantidad de cambios posibles en la estructura de la misma, dada la existencia de datos que se desean conservar.
* Es necesario la separación del proyecto pues se encuentra en estado monolítico, o sea, la interfaz gráfica y la lógica se encuentran muy acopladas, lo que puede ocasionar problemas de escalabilidad.
* Se decide migrar la interfaz gráfica para solucionar varias insuficiencias que afectan la usabilidad de la aplicación. Para ello se propone utilizar el *framework* de *front-end* Vue.js 3 por su baja curva de aprendizaje, y por las ventajas que brinda, como por ejemplo, la reactividad, para la validación en tiempo real de las respuestas en las ejecuciones de los test psicológicos.
* Se decide desarrollar una API para mejorar el rendimiento y escalabilidad de la aplicación. Para ello se propone utilizar el *framework* de *back-end* *NestJS*, por su rendimiento considerable en escenarios de alta concurrencia, ecosistema de bibliotecas amplio y comunidad activa.

# Capítulo 2: Propuesta de solución

## 2.1 Introducción

En este capítulo se expone el proceso de captura de requisitos y diseño de software realizado para el desarrollo de la solución. Luego, se explican las modificaciones realizadas al diseño de base de datos, las cuales permiten la separación lógica de la información acorde al cliente que la solicita. Se presenta la nueva interfaz visual, así como la arquitectura del proyecto de *front-end* y *back-end*. Por último, se presentan las pautas utilizadas en el diseño de las interfaces.

## 2.2 Requisitos funcionales

A partir de técnicas como entrevistas, arqueología de documentos y software, se obtuvieron los siguientes requisitos funcionales. Ya han sido implementados en el sistema los primeros 23 requisitos (hasta el R23, incluyéndolo); los restantes son objetivo de versiones futuras [16, 18]:

R1. El sistema debe permitir al administrador crear grupos de usuarios.

R2. El sistema debe permitir al administrador modificar grupos de usuarios.

R3. El sistema debe permitir al administrador eliminar grupos de usuarios.

R4. El sistema debe permitir al administrador crear usuarios.

R5. El sistema debe permitir al administrador modificar la información de los usuarios.

R6. El sistema debe permitir al administrador eliminar usuarios.

R7. El sistema debe permitir al proveedor del sistema crear clientes.

R8. El sistema debe permitir al proveedor del sistema eliminar clientes.

R9. El sistema debe permitir al administrador, designar a los usuarios en su grupo correspondiente.

R10. El sistema debe permitir al cliente registrarse como un cliente del sistema.

R11. El sistema debe permitir al cliente crear grupos de usuarios dentro de sus grupos posibles.

R12. El sistema debe permitir al cliente modificar sus grupos de usuarios.

R13. El sistema debe permitir al cliente eliminar sus grupos de usuarios.

R14. El sistema debe permitir al cliente crear usuarios dentro de sus grupos posibles.

R15. El sistema debe permitir al cliente modificar la información de los usuarios dentro de sus grupos posibles.

R16. El sistema debe permitir al cliente eliminar usuarios dentro de sus grupos posibles.

R17. El sistema debe permitir al cliente, designar a los usuarios en su grupo correspondiente, dentro de sus grupos posibles.

R18. El sistema debe permitir al analista establecer los test a realizar para un grupo de usuarios.

R19. El sistema debe permitir al analista gestionar un test psicológico.

R20. El sistema debe permitir al ejecutor realizar el test.

R21. El sistema debe permitir al ejecutor obtener los resultados de un test que haya realizado.

R22. El sistema debe permitir al analista obtener los resultados por test de un usuario.

R23. El sistema debe permitir al usuario autenticarse.

R24. El sistema debe permitir al usuario cambiar su contraseña.

R25. El sistema debe permitir al analista obtener resultados de test por grupos de usuarios.

R26. El sistema debe permitir al administrador modificar roles funcionales.

R27. El sistema debe permitir al administrador eliminar roles funcionales.

R28. El sistema debe permitir al analista visualizar en un grupo los líderes y las incompatibilidades.

R29. El sistema debe permitir al administrador el registro de los roles funcionales que va a manejar la organización.

R30. El sistema debe permitir seleccionar a cada usuario sus roles funcionales preferidos y evitados.

R31. El sistema debe permitir a cada usuario seleccionar entre las personas que considera líderes dentro de su grupo.

R32. El sistema debe permitir a cada usuario seleccionar las personas de su grupo con las que se considera incompatible.

## 2.3 Requisitos de calidad

El sistema debería cumplir los siguientes requisitos de calidad:

**Seguridad:**

* Solo los usuarios autorizados (analistas) pueden acceder a la información de la ejecución de los test de otros usuarios.
* La información de los grupos de usuarios correspondientes a cada cliente debe estar separada, y no puede ser visible para los otros clientes.
* El sistema debe registrar trazas de las acciones de lectura, inserción, actualización y eliminación de todos los datos de la aplicación para poder ser auditados si es posible.

**Usabilidad:**

* Ofrecer los componentes adecuados para dar respuesta a cada tipo de pregunta en la ejecución de un test.
* Ofrecer filtros y buscadores para facilitar a los analistas la gestión de los datos.
* Ofrecer componentes de inserción de datos adecuados para cada tipo de dato en los formularios.
* Mostrar mensajes de información en los procesos de creación de test psicológicos para orientar al analista.

**Fiabilidad:**

* Notificar al usuario en cada una de las operaciones a realizar el estado de la misma, ya sea completada satisfactoriamente o fallida.

**Interfaz externa:**

* Uso de colores que transmitan calma al usuario durante la ejecución de los test.
* Uso de componentes adecuados para las respuestas en la ejecución de los test psicológicos.
* Mostrar el tiempo de ejecución de los test solo cuando el usuario lo requiera para evitar estrés innecesario.

**Legales:**

* Al cliente se le notificará a través de términos y condiciones de que su información será utilizada sólo con fines académicos y profesionales.

**Rendimiento:**

* El sistema debe ser capaz de responder a escenarios de alta concurrencia como lo es la ejecución simultánea de los test.

**Restricciones:**

El terminal donde se debe desplegar la aplicación debe poseer:

* MariaDB 10.6.5
* Node 20
* Debian Linux 10
* Redis 7.4.2
* Memoria RAM: 16 Gb (para la alta concurrencia en ejecuciones de test simultáneas)
* Almacenamiento: 100 Gb (para almacenar test con imágenes)

## 2.4 Diagrama de casos de uso del sistema

A continuación, la Figura 4, donde se representa el diagrama de los casos de uso del sistema:

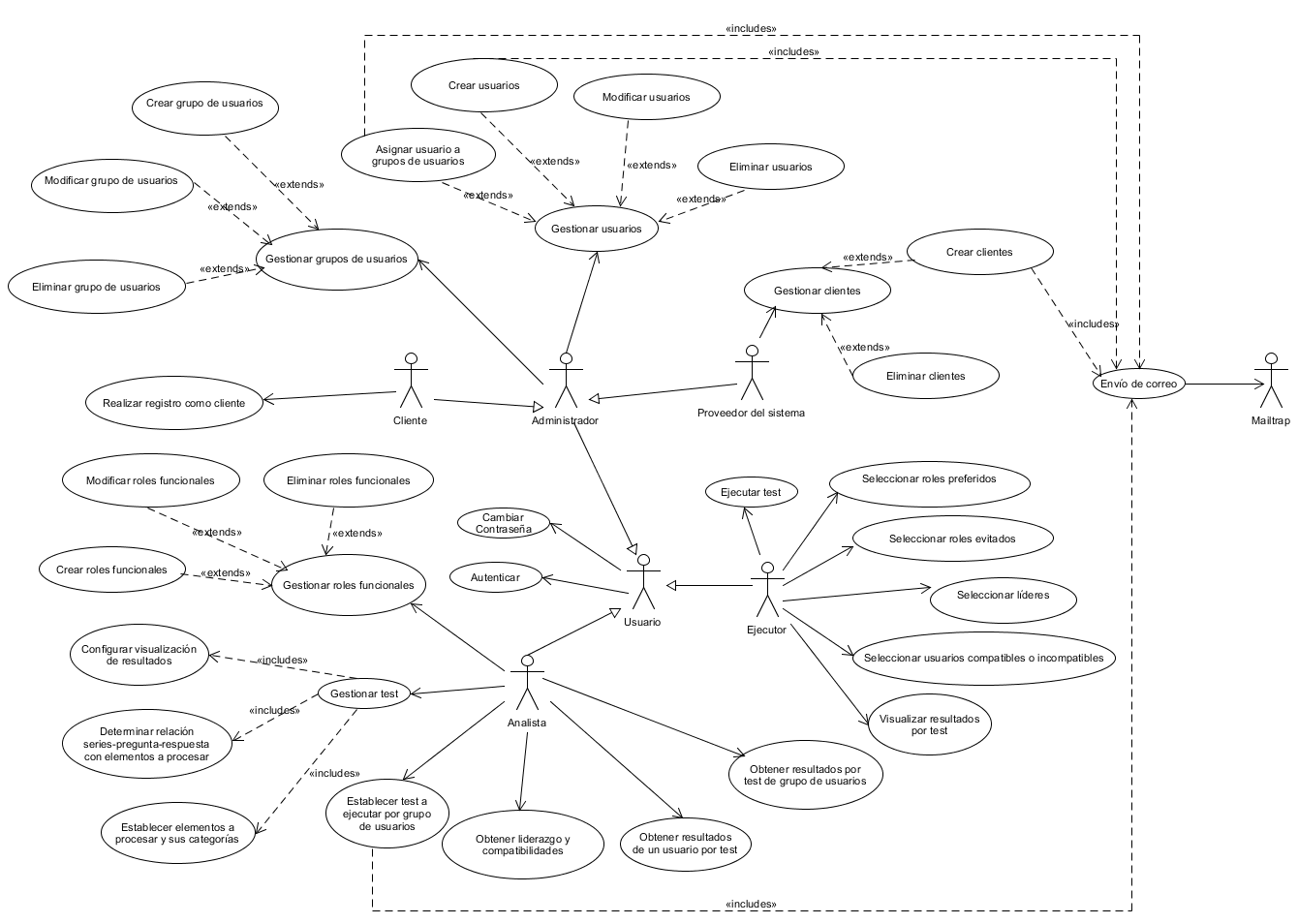


Figura 4: Diagrama de casos de uso del sistema

Los actores del sistema son los siguientes:

* **Usuario:** Representa la entidad que interactúa con el sistema. Es un actor genérico del cuál heredan el resto de los actores, ya que todo usuario que se conecte al sistema, con independencia del rol que juegue, debe autenticarse y puede cambiar su contraseña.
* **Analista:** Representa al usuario especializado en el negocio, que se encarga de la creación de los test psicológicos, asignación de roles funcionales y la ejecución de análisis sobre los resultados de los tests y, sobre el liderazgo y las incompatibilidades en grupos de usuarios.
* **Ejecutor:** Representa al usuario capaz de ejecutar un test con el fin de evaluarse. Provee los datos necesarios para que el analista pueda realizar análisis con los mismos, y pueda consultar sus resultados.
* **Administrador:** Representa al usuario responsable de la administración de los datos relacionados con los usuarios, clientes y grupos de usuarios de la aplicación.
* **Proveedor del sistema:** Es un tipo de administrador, que, además de poder controlar toda la información de grupos de usuarios y test, puede gestionar los clientes del servicio.
* **Cliente:** Representa al usuario que contrata el servicio, responsable de la administración de los datos relacionados con los usuarios y grupos de usuarios visibles al mismo.
* **Mailtrap:** Representa al servidor de correos externos que se utiliza para enviar correos a los usuarios para notificar acciones como su registro en el sistema.

## 2.5 Arquitectura de la solución propuesta

La herramienta en su estado actual sigue una arquitectura monolítica de modelo vista controlador (MVC). Todo el código de la interfaz visual y lógica se encuentra junto en el mismo proyecto, aunque separado en distintas capas estrechamente relacionadas [16].

Al encontrarse en un estado monolítico, presenta diversas desventajas para la migración a un modelo SaaS, como lo es, principalmente, la dificultad para su escalabilidad (aspecto de suma importancia para el modelo de software como servicio). En este sentido el mantenimiento de los componentes, que por lo general se encuentran acoplados en una única estructura, se dificulta demasiado y por tanto, se sugiere la separación de los mismos para que puedan escalar individualmente [26].

Existe como alternativa la arquitectura de microservicios, la cual se caracteriza por dividirse en varios servicios independientes que se ejecutan como unidades autónomas. Estos no dependen entre sí y pueden ser desarrollados, desplegados y escalados de forma independiente. Cada servicio, por lo general, posee su propia base de datos, lenguaje de programación y *framework*. Una de las ventajas principales que ofrece es que el fallo de un servicio no afecta al resto del sistema. A pesar de que la alternativa parece ser adecuada, el *back-end* del sistema ya cumple con las necesidades del negocio, y lo que se requiere es separar es la creación de una nueva aplicación *front-end* que consuma de la misma. Además, los microservicios no se recomiendan implementar para sistemas con pocas funcionalidades y una base de código relativamente pequeña. Las comunicaciones entre servicios pueden generar una latencia mayor, lo que puede ser un problema en sistemas con requisitos de rendimiento estrictos, y el desarrollo y mantenimiento de microservicios puede ser costoso y complejo [42].

La arquitectura Cliente-Servidor es un modelo más sencillo que la arquitectura de microservicios, en el que los clientes (como aplicaciones o usuarios) solicitan servicios a un servidor centralizado (también conocido como host). Los clientes envían solicitudes al servidor, que responde proporcionando los servicios solicitados. Entre sus ventajas con respecto al proyecto en cuestión [43]:

* Los clientes no necesitan conocer detalles específicos del sistema (software o hardware), aspecto fundamental para solucionar la problemática.
* Los clientes suelen estar en estaciones de trabajo o computadoras personales, mientras que el servidor está en una máquina más potente en la red. Esta máquina es la que se encargará de almacenar los datos centralizados del negocio en una sola base de datos, lo cual es requerido para realizar estudios y análisis científicos y académicos sobre los resultados de los test psicológicos.

El *back-end* se encuentra conformado por una serie de controladores que se comunican directamente con la lógica que gestiona las visuales. La nueva interfaz visual se debe comunicar con el *back-end* a través de una *Api* mediante peticiones *http.*

La arquitectura *back-end* sigue la estructura de controladores, encargados de definir las rutas por las que transita la información siempre y cuando las credenciales de autorización sean correctas, servicios, donde se maneja la lógica de negocio, y los modelos, que representan las entidades en base de datos para el manejo de la información. De esta forma la escalabilidad del *back-end* resulta bastante sencilla, al mantener una separación de las distintas capas.

Por las razones expuestas, se decide como la arquitectura predominante “Cliente-Servidor”.

El sistema se dividirá en múltiples capas (o niveles) independientes, cada una con responsabilidades y funciones específicas. Esta separación en capas permite una mejor organización, modularidad y escalabilidad del sistema. Cada capa es independiente de las demás, lo que facilita el desarrollo, mantenimiento y actualización del sistema [44].

Las capas generalmente pueden reutilizarse en otros sistemas, lo que reduce el tiempo de desarrollo. Se pueden agregar o eliminar las mismas fácilmente para satisfacer las necesidades cambiantes del sistema. Cada capa se encarga de una función específica, lo que mejora la legibilidad y el mantenimiento del código. Las diferentes capas pueden usar diferentes tecnologías, lo que permite flexibilidad y adaptación [44].

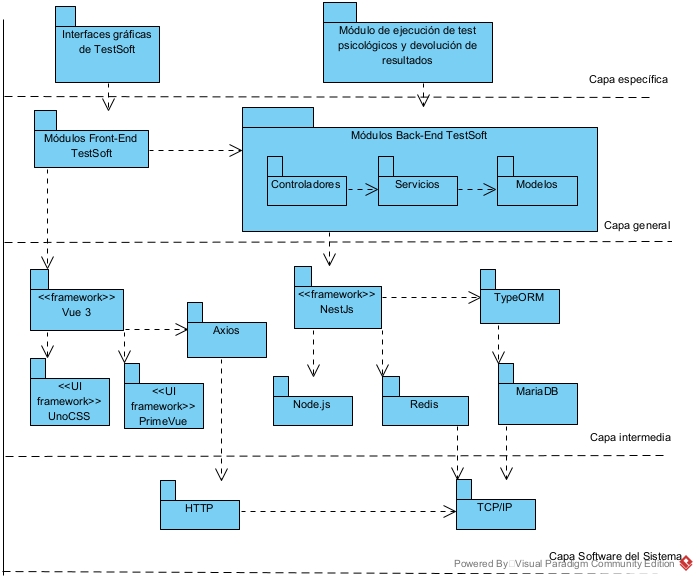


Figura 5: Representación del patrón N-Capas en el proyecto.

El sistema, como se observa en la Figura 5, se divide en varias capas definidas por reutilización, con el objetivo de al escalar el sistema con acciones como añadir nuevos tipos de test y funcionalidades a la aplicación, la arquitectura facilite dichas acciones y permita mantener una consistencia en la estructura del proyecto.

* **Capa específica**: Constituye la capa del sistema que contiene elementos no reutilizables y que se relacionan fuertemente con las reglas del negocio de la aplicación. En esta capa se encuentran:
  + Interfaces gráficas de TestSoft: Establece la forma de interacción con el usuario, la recepción y validación de los datos. Aquí se encuentran las interfaces gráficas de la aplicación pues son los elementos más específicos del sistema.
  + Módulo de ejecución de test psicológicos y devolución de resultados: Agrupa la ejecución de test psicológicos y el cálculo de los resultados, lógica que es especifica del sistema.
* **Capa general**: Llevan a cabo la lógica del negocio que puede ser reutilizada en otros proyectos. Maneja la comunicación entre las aplicaciones *front-end* y *back-end,* einteractúa con la información en base de datos. En esta capa se encuentran:
  + Módulos Front-End: Agrupa la lógica de conexión con la API de cada módulo.
  + Módulos Back-End: Agrupa la lógica de negocio de cada módulo y sus conexiones a la base de datos. Dentro de los módulos se encuentran los controladores (encargados de procesar las peticiones), los servicios (contienen la lógica de negocio) y los modelos (representaciones de las entidades de la aplicación).
* **Capa intermedia**: Establece las bibliotecas, *frameworks* y software que requieren las capas superiores para funcionar. Agrupa el *framework* de *front-end* Vue.js, el cual consume de los *frameworks UI UnoCSS* (biblioteca de estilos) y *Primevue* (biblioteca de componentes), y la biblioteca *Axios*, encargada de realizar las peticiones. Por otra parte, se encuentra *NestJs*, que consume de *Node.js*, y de *TypeORM* para poder realizar las consultas a la base de datos, que se encuentra desarrollada en *MariaDB*.
* **Capa Software del sistema**: Establece los elementos que utiliza la capa intermedia para poder realizar acciones como la comunicación, para la cual se utilizan los protocolos HTTP y TCP/IP.

## 2.6 Patrones de diseño utilizados

Para resolver ciertas problemáticas se utilizaron un conjunto de patrones de diseño, los cuales, además de brindar una solución a las mismas, mejoran la calidad del código y su escalabilidad.

### 2.6.1 *Strategy*

La clase *TestExecution* posee una diversidad de preguntas, y cada una de ellas requiere una validación en dependencia de su tipo. El patrón “*Strategy*” es usado porque cada tipo de pregunta requiere una implementación en específico del método *validateQuestion*, como se puede observar en la Figura 6. Para ello se crean clases que extiendan de *Question* e implementen el método en cuestión a su forma*.* Este patrón se aprovecha con el objetivo de que sea más sencillo el proceso de añadir nuevos tipos de preguntas.

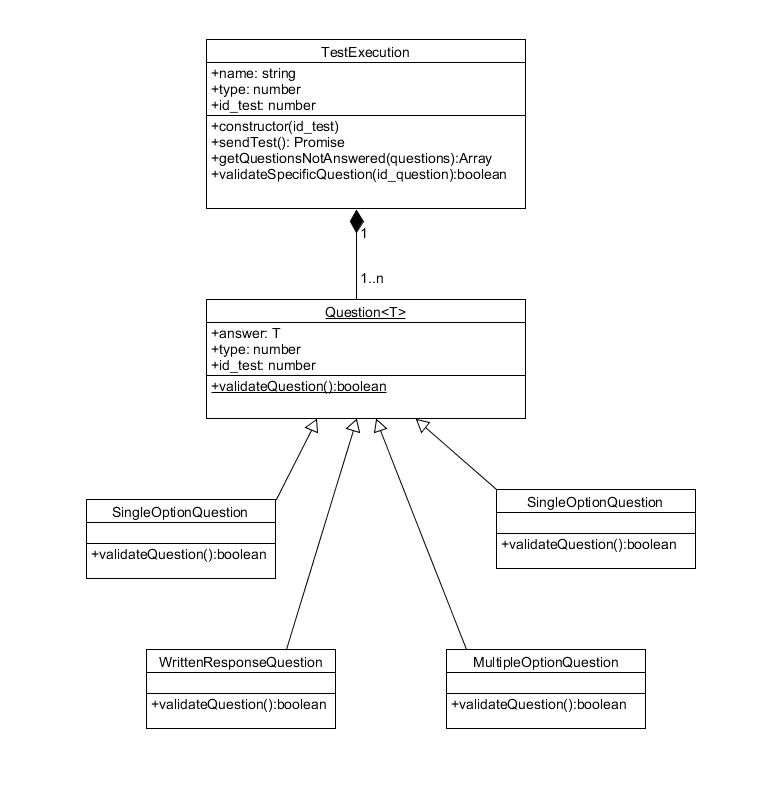


Figura 6: Representación del patrón *Strategy* en el proyecto.

### 2.6.2 *Builder*

Este patrón, representado en la Figura 7, se selecciona con el objetivo de facilitar la implementación de la creación de un test. Este proceso es complejo pues requiere diversos pasos que serán efectuados en diversos momentos e interfaces gráficas, por lo que es de suma utilidad el patrón para construir el test a lo largo del flujo. Se utiliza una clase *TestBuilder* que se encarga de almacenar una instancia de *Test*, y a través de diversos métodos, se construye progresivamente [1].

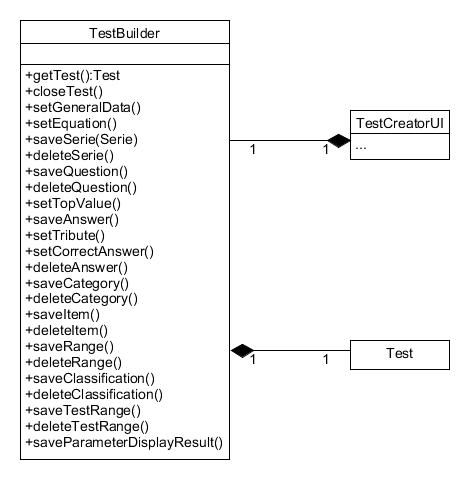


Figura 7: Representación del patrón *Builder* en el proyecto.

### 2.6.3 *Facade*

Usado para abstraer al cliente del complejo procesamiento de un test y la obtención de sus resultados dada la cantidad de entidades involucradas y operaciones que es necesario realizar. De esta manera se simplifica el inicio de tal proceso y la obtención del resultado del mismo.

Como se observa en la Figura 8, el proceso de ejecución de un test es complejo y requiere la intervención de distintos métodos y modelos, por lo que se hace necesario abstraer al cliente de toda esta maquinaria. Para esto el patrón “*Facade*” es de suma utilidad, estableciendo una vía para la inicialización del proceso y la obtención de la información resultante del mismo.

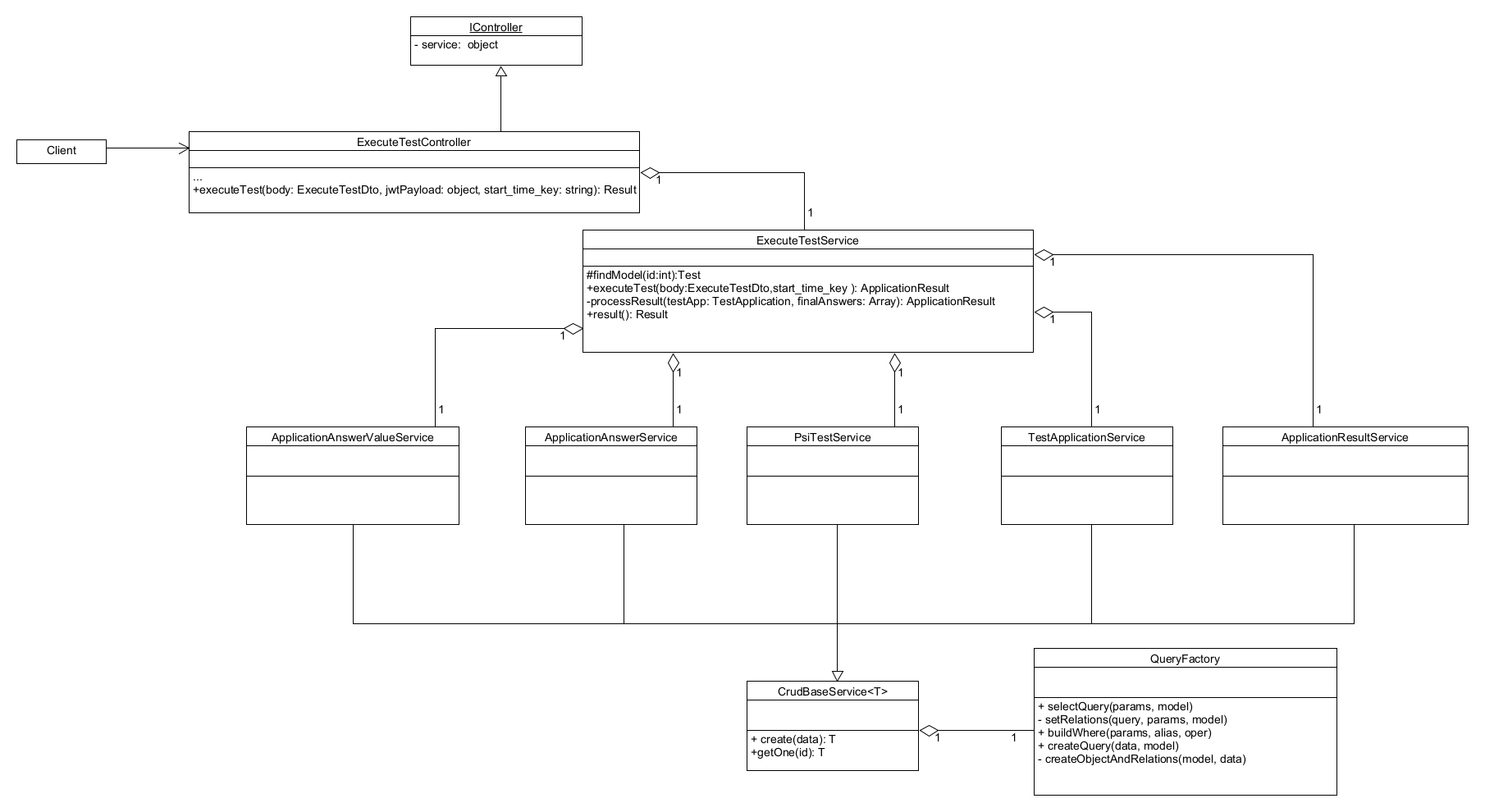


Figura 8: Representación del patrón *Facade* en el proyecto.

## 2.7 Propuesta de rediseño de base de datos

A continuación, se explican los cambios sugeridos al modelo de base de datos para la posterior migración de la herramienta a un Software como servicio.

La base de datos contiene al test, incluye los principales parámetros que lo definen, como son: el nombre, la descripción de los objetivos que se buscan alcanzar con su aplicación, el tiempo máximo de duración para completarlo y el intervalo de tiempo en el que debe repetirse su ejecución (en caso de que los resultados puedan variar con el tiempo). Además, se especifica si es posible navegar entre las diferentes series que componen el test.

Como se mencionó anteriormente, los test pueden ser de dos tipos: psicométricos o de personalidad. En el caso de los test psicométricos, su procesamiento generalmente requiere una fórmula que toma en cuenta las respuestas correctas, incorrectas, omitidas y el total de preguntas. Sin embargo, cada test puede tener un método de procesamiento particular. Por otro lado, en los test de personalidad, es fundamental que el usuario responda todas las preguntas, ya que omitir alguna puede llevar a resultados incorrectos. Esta es una de las principales diferencias entre ambos tipos de test.

En los test psicométricos, una pregunta no respondida puede deberse a falta de conocimiento o tiempo, y además es posible dar una respuesta incorrecta. En cambio, en los test de personalidad no existen respuestas correctas o incorrectas; cada respuesta aporta información valiosa para identificar aspectos de la personalidad, por lo que es crucial verificar que el test esté completo.

Cada test está compuesto por al menos una serie de preguntas, aunque generalmente incluye varias. Cada serie tiene un nombre y un tiempo de duración definido, ya que en algunos casos se establece un límite de tiempo para completar cada serie.

Cada serie, a su vez, está formada por preguntas, y para cada pregunta se conoce su enunciado y su tipo. Además, cada pregunta puede tener una o varias respuestas posibles.

Los usuarios que responden los test poseen datos como: número de identidad, nombre(s), apellido(s), sexo, nombre de usuario, contraseña y correo electrónico. Estos a su vez poseen determinados roles, los cuales son desempeñados en los grupos a los que pertenecen. Estos grupos a su vez se encuentran organizados en una estructura jerárquica, pues cada uno conoce el grupo al que pertenece, también llamado grupo padre.

Cada usuario responde los test de manera única. Esta información es clave para analizar los resultados y obtener conclusiones relevantes.

En vista al objetivo final, o sea, llevar la herramienta a SaaS, se propone una serie de cambios al modelo físico de la base de datos. Primeramente, remarcar que se decide mantener el modelo de base de datos única (Base de datos compartida y un esquema), y se procede a separar la información de los *tenant* de forma lógica. Con ello se conserva la base de datos existentes y su estructura no se ve tan afectada ya que es necesario para el proveedor, la conservación de los datos ya existentes en la misma para la realización de análisis e investigaciones.

Durante el análisis del estado actual del diseño de la base de datos se detectó que, aunque hay que realizar ajustes, estos no son de gran envergadura. La forma en que se concibió la herramienta, organiza a los usuarios por grupos, lo que posibilita que la mayor parte de los cambios se puedan hacer de forma lógica sin necesidad de reestructurar la base de datos.

Aun así, uno de los primeros aspectos a tener en cuenta es el cambio de los roles dentro del sistema. Esto no está directamente relacionado al diseño de base de datos pues estos roles y permisos se encuentran definidos en la lógica del sistema. Dejan de ser Estudiante, Especialista y Analista, para pasar a ser, **Ejecutor**, **Analista**, **Administrador, Cliente y Proveedor**.

Este primer cambio responde a que se pretende que el sistema no aplique sólo para el sector educacional, por lo que los roles como se concibieron en un inicio, limitan el uso de la herramienta.

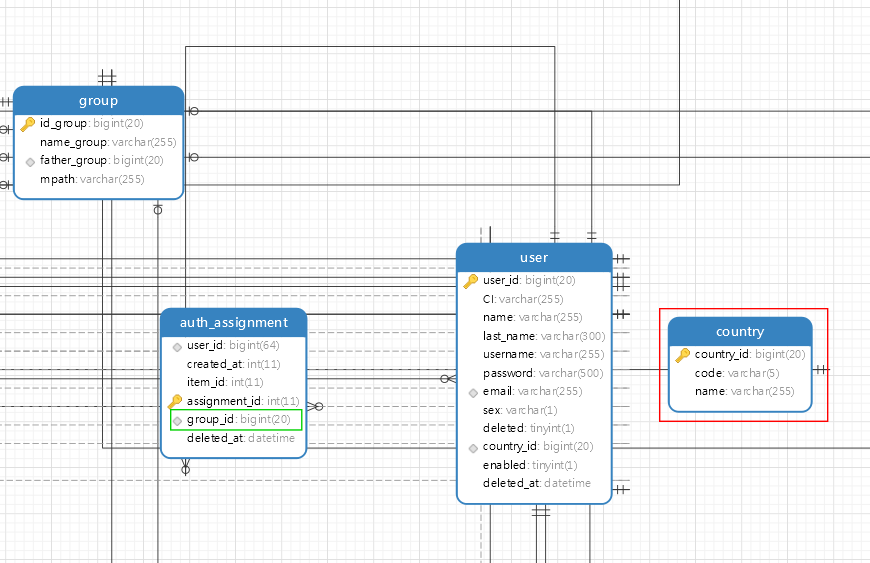
****

Figura 9**:** Fragmento del modelo físico de base de datos donde se muestran los cambios en las asignaciones de usuarios en grupos

Se planteó la posibilidad de la presencia de un usuario en más de un grupo con un rol distinto en cada uno. Para modelar esto se establece una relación entre *auth\_assignment* y *group* manejando así múltiples asignaciones de un usuario en varios grupos con un rol diferente. Esta modificación se señala en color verde en la Figura 9.

Con el objetivo de llegar a ser una herramienta de uso internacional se añade un campo ***country\_id***y un nomenclador ***Country***, que sería una tabla de países, como se puede observar en rojo en la Figura 9.

Algunos test como MBTI precisan lo que se debe hacer cuando existan empates en las puntuaciones de los elementos de una categoría. En la versión anterior, aunque para MBTI funcionaba, este tipo de comportamientos no estaba generalizado ni era posible configurarlo. Para resolver este problema se crea un campo para registrar la prioridad de un *item* en la categoría y poder manejar los empates en los resultados de un test.

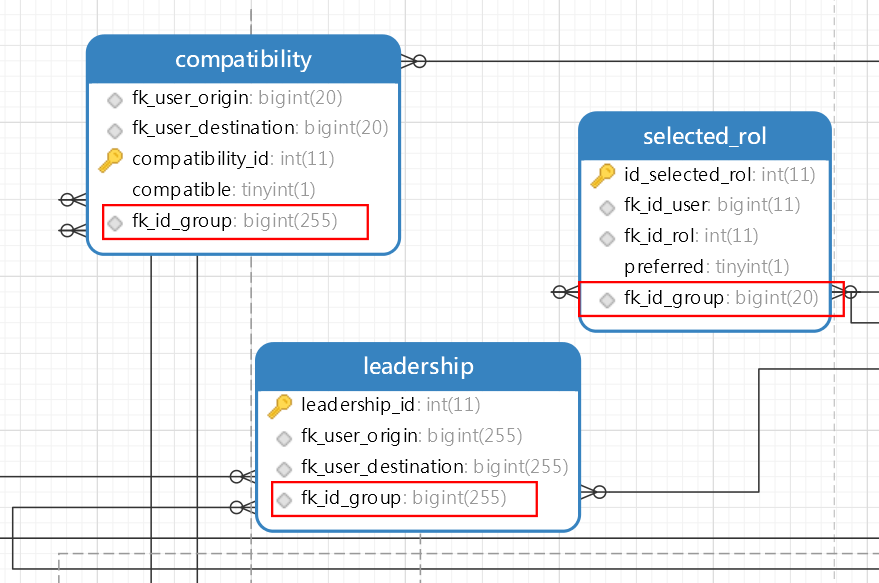


Figura 10: Fragmento del modelo físico de la base de datos. En rojo, los nuevos campos añadidos en relación al grupo donde se hizo la selección.

Se incluyeron en las tablas *compatibility*, *leadership* y *selected\_rol*, el atributo *fk\_id\_group*, como se puede observar en la Figura 10, con el objetivo de guardar el grupo donde un usuario hizo la elección de liderazgo, compatibilidad o preferencia del rol, y con ello separar lógicamente la información, para evitar que sea consultada por otro usuario no autorizado a ver ese grupo en específico.

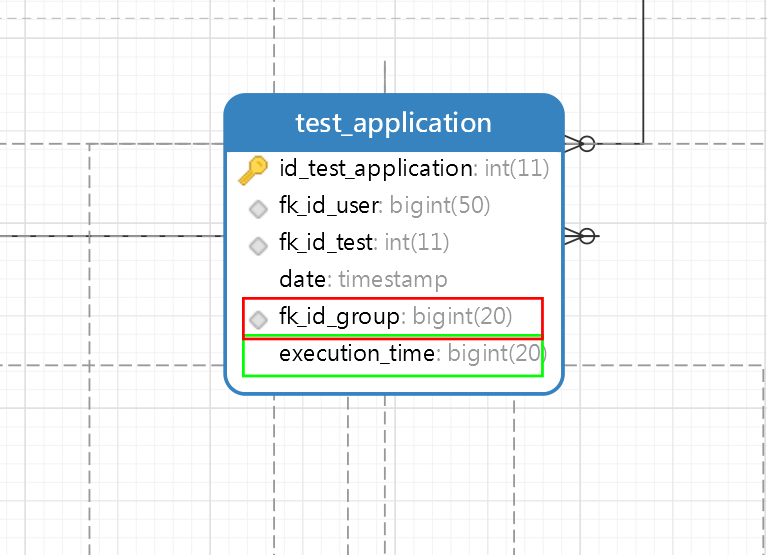


Figura 11: Fragmento del modelo físico de la base de datos donde se muestran los nuevos atributos de la tabla *test\_application*.

Se incluyó en la tabla *test\_application* el atributo *fk\_id\_group*, como se puede observar en rojo en la Figura 11, con el objetivo de guardar el grupo donde un usuario hizo realizó una ejecución de test determinada, y con ello evitar que la información sea consultada por otro usuario no autorizado a ver ese grupo en específico, similar a las tablas de la Figura 10.

Se creó un nuevo atributo *execution\_time*, que indica el tiempo que se tardó el usuario en realizar la ejecución de un test determinando, como se puede observar en verde en la Figura 11. Este aspecto puede otorgar información valiosa para el posterior análisis del rendimiento de los usuarios examinados [45].

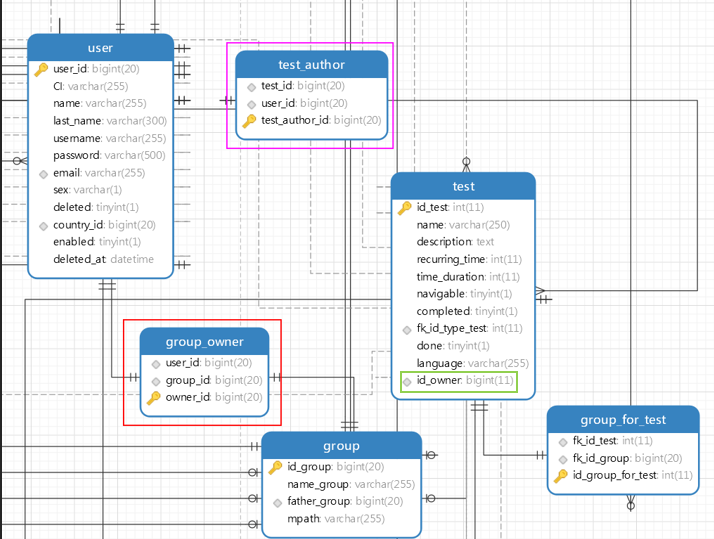


Figura 12: Fragmento del modelo físico de la base de datos donde se muestran las nuevas relaciones entre test, grupos y usuarios.

Se crea la tabla *group\_owner* para establecer una relación de muchos a muchos entre usuarios y grupos. Estos serían los usuarios que solicitarían el servicio, y crearían un grupo que sería la raíz de la jerarquía de grupos en el sistema para este usuario. Esta modificación se señala en color rojo en la Figura 12.

Se crea el atributo *id\_owner* en la tabla *test* para indicar el grupo dueño del test, como se puede observar en la Figura 12 en color verde, lo que ayuda a que los analistas que formen parte de este grupo o de sus grupos hijos, puedan asignarlos a los usuarios de esos grupos.

Se crea la tabla *test\_author* para indicar los usuarios autores de un test determinado, como se puede observar en la Figura 12 en color rosado, lo que posibilita que solo ellos sean los responsables de la edición de ese test en cuestión.

Se decide sustituir la llave primaria de la tabla ***User***. Actualmente, la llave primaria es el número de carné de la persona con *bigint* como tipo de dato, lo cual provoca inconsistencia, un error en la información si la persona nació a partir del año 2000, ya que el 0 del inicio del número se pierde. La falta de generalidad por parte de la llave primaria llega a ser un problema, sobre todo con vistas a internacionalizar la herramienta. Remarcar que si en un determinado momento, el formato del carné llegase a cambiar, podría traer problemas a la hora de actualizar la información en la base de datos. Por ende, se sugiere sustituir la llave primaria por un entero autoincremental en la tabla ***User***.

Otro cambio sugerido es la unión de las tablas de *student\_group*y *specialist\_group*, porque ambas tablas son idénticas y de esta forma se evita la comprobación del tipo de usuario cada vez que se desea obtener alguna información de su grupo, una vez más quedaría de parte de la lógica de la herramienta identificar el tipo de usuario al momento de procesar los datos para los reportes o los permisos. Se eliminaría el campo de *group\_type* de la tabla *group\_for\_test,* ya que su propósito era indicar la tabla de grupo que se debía consultar. Al existir una única tabla de grupos este campo queda obsoleto.

## 2.8 Migración de la interfaz gráfica

Dados los problemas de usabilidad que presenta la interfaz de usuario de la versión actual del software, en especial, la falta de una mayor separación de lógica e interfaz, hecho que dificulta la escalabilidad y mantenimiento, se decide llevar a cabo una migración de la interfaz visual.

Con las tecnologías del *framework* *Vue.js 3* y *Typescript* se desarrolla una nueva interfaz visual separada en su totalidad de la lógica de negocio, lo que la convierte en una *Single Page Application* (SPA). Esta consume la información realizando peticiones *http* a una *Api rest-full* desarrollada en *NestJS.*

### 2.8.1 Arquitectura del proyecto de *front-end*

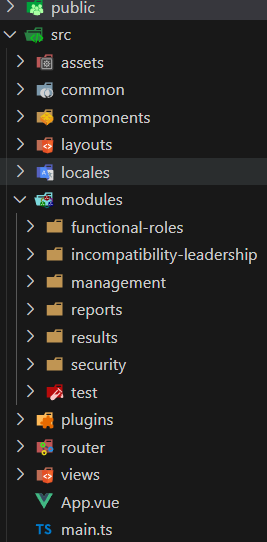
******

Figura 13: Jerarquía de directorios en el proyecto *front-end*

El proyecto de *front-end* presenta la siguiente arquitectura, la cual se puede apreciar en la Figura 13. En la carpeta raíz del proyecto se encuentran los directorios:

* ***public***: Carpeta donde se guardan aquellos archivos de uso en toda la aplicación.
* ***src***: Almacena el código fuente del proyecto desarrollado.

El resto de archivos en su mayoría son ficheros de configuración creados por defecto cuando se inicia un nuevo proyecto de ***Vue.js 3*** con la herramienta ***Vite***.

En la carpeta ***src*** el proyecto se divide en varios directorios por responsabilidad:

* ***assets***: Guarda los recursos como fuentes, iconos e imágenes.
* ***components***: Componentes visuales comunes al proyecto.
* ***layouts***: Plantillas bases de las distintas vistas del proyecto.
* ***router***: Lógica que maneja la navegación del usuario por las distintas vistas.
* ***plugins***: Archivos de configuración de algunas dependencias.
* ***common***: Funcionalidades y útiles comunes al proyecto.
* ***views***: Vistas particulares del proyecto.

En la carpeta ***modules*** se encuentran distintos módulos separados por responsabilidad con sus respectivas lógicas. Cada módulo de forma general posee clases, vistas, esquemas de validación, ***composables*** y ***stores*** en caso de llevarlos, como sucede con el módulo de seguridad. Las vistas son aquellas que corresponden a la lógica encapsulada en el módulo. Por ejemplo, las vistas del módulo ***test*** son las de la aplicación de un test psicológico.

En este caso hay tres módulos:

* ***security***: Posee la lógica relacionada a la seguridad de la aplicación. En él se gestiona la información del usuario, así como su autenticación y autorización.
* ***functional\_roles***: Todo aquello relacionado a la preferencia de roles funcionales.
* ***Incompatibility-leadership***: Todo aquello relacionado a la selección de usuarios lideres, compatibles e incompatibles.
* ***test***: Todo aquello relacionado a la aplicación de un determinado test, desde realizar la petición del test a realizar, validar el test, hasta enviar la información de las respuestas para ser procesado en el *back-end*.
* ***management***: Todo aquello relacionado a la gestión de los datos de la aplicación, ya sea usuarios, grupos y test psicológicos.
* ***results***: Posee la lógica para la visualización de los resultados de un test y de todos aquellos test que haya realizado el usuario.
* ***reports***: Todo lo relacionado a los diferentes reportes que brinda la aplicación.

Para la implementación de las vistas de un test, se decidió tratar al test en sí como un contenedor genérico de preguntas de distintos tipos. Cada pregunta es un componente diferente en sí, y en dependencia de los tipos de preguntas que posea un test, se *renderiza* un componente u otro.

De esta forma es más versátil la aplicación, ya que, sin importar el test, con conocer los tipos de preguntas que posee es muy sencilla la implementación para mostrarlo. Además, permite la creación de nuevos test sin la necesidad de implementar nuevo código. De ser necesario un nuevo tipo de pregunta, implementarlo es muy sencillo porque conlleva tan solo crear el componente y agregarlo como posible tipo de pregunta a seleccionar.

### 2.8.2 Consideraciones en el diseño de la interfaz

Para la implementación de la interfaz se utilizaron principalmente componentes de la biblioteca de componentes para ***Vue.js***, ***PrimeVue***.

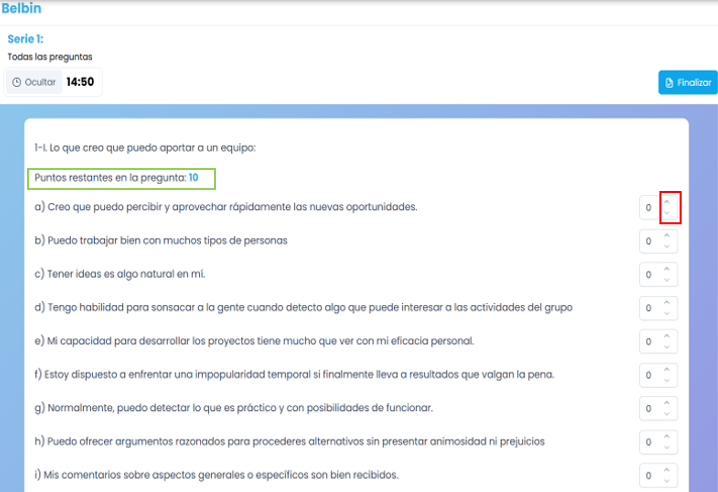


Figura 14: Fragmento de la interfaz de ejecución del test de Belbin en la nueva versión del software.

Como se puede apreciar observar en la Figura 14, se añadieron botones para el incremento y decremento de los valores de las respuestas, en rojo, y se indica en la parte superior izquierda del componente, en verde, los puntos restantes. Además, se introdujo, un mensaje de información que muestra al usuario los puntos restantes en la pregunta que se encuentra respondiendo en el momento, como se observa en la Figura 15, en la sección superior izquierda, en color rojo. Esta característica se introdujo para el caso en que la cantidad de puntos restantes no sea visible por el *scroll*.

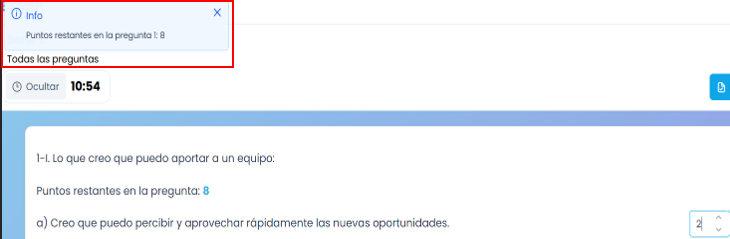


Figura 15: Fragmento de la interfaz de ejecución del test de Belbin (muestra los puntos restantes de la pregunta al asignar puntos a una de sus respuestas).

Las preguntas presentan el mismo estilo que en la versión inicial de la interfaz, ya que la forma de responder, ya sea por *checkbox*, radio botones, entradas de números o texto, es intrínseco al tipo de pregunta del test y a cómo se diseñó el test en primer lugar.

Se realizaron mejoras en el diseño *responsive* de la interfaz visual. La versión anterior presentaba problemas a la hora de ser visualizada en dispositivos móviles, deficiencia resuelta en la nueva versión.

La Figura 16 muestra los mensajes de error a la hora de enviar y guardar las respuestas de un test, los cuales revelan mayor información, como lo es la serie y pregunta que presenta problemas en la respuesta. Además, como se puede ver en la Figura 17, las preguntas incompletas serán señaladas en rojo hasta que sean respondidas, funcionalidad que fue implementada gracias a la reactividad que ofrece el *framework* seleccionado.

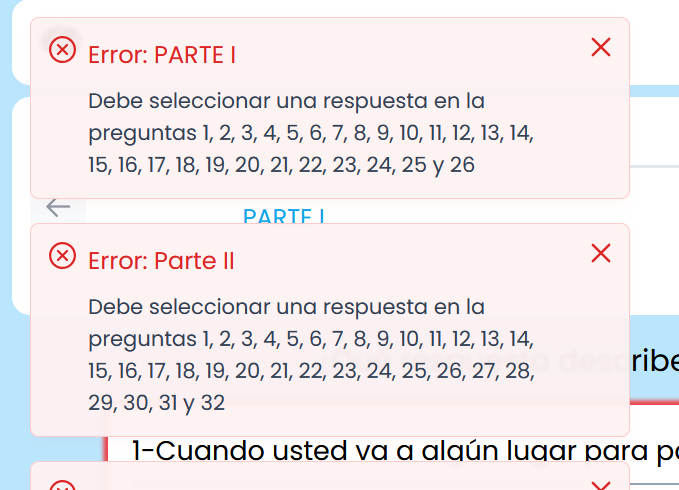


Figura 16: Fragmento de interfaz de ejecución del test MBTI (muestra mensajes indicando preguntas sin responder).

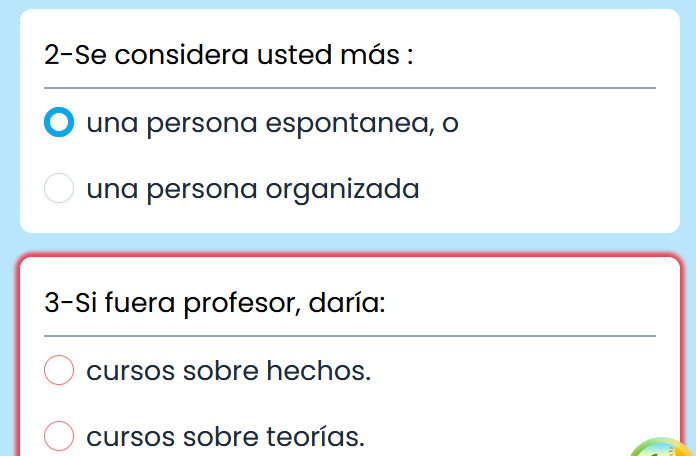


Figura 17: Fragmento de interfaz del test MBTI (se señala en rojo la pregunta sin responder)

Además, se implementó una funcionalidad que permite guardar las respuestas y el tiempo restante de una ejecución de test temporalmente en el almacenamiento local del navegador. Esto permite que al refrescar el navegador no se reinicie el tiempo de ejecución del test, y poder conservar las respuestas, en caso de que se cierre el navegador, por ejemplo, y no tener que repetir el llenado de las respuestas.

Con respecto al proceso de creación del test, se implementaron ciertas mejoras:

* La creación de las series y preguntas se diseñó de forma tal que fueran mostradas de manera similar a como se vería ese test en ejecución, como se puede observar en la Figura 18 . Esto permite al analista tener una visión más intuitiva de cómo sería la ejecución del test que está creando.

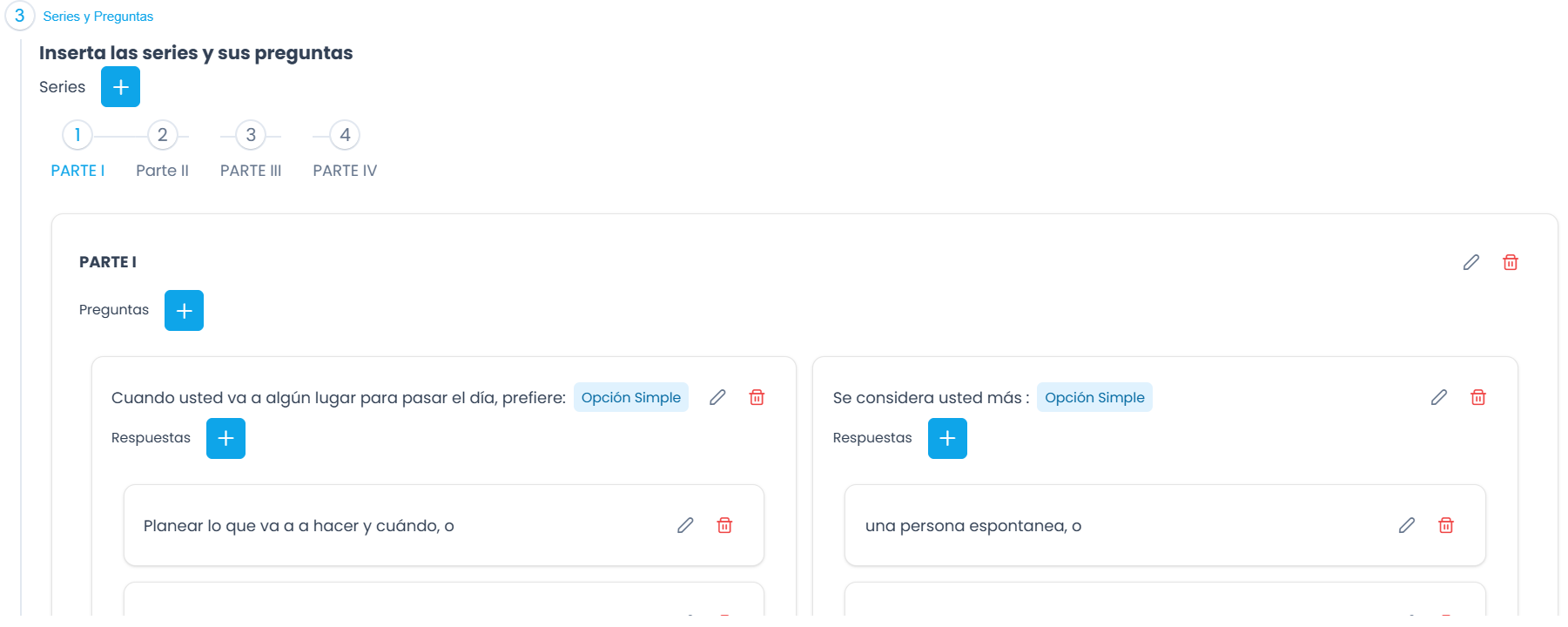


Figura 18: Proceso de creación de series y preguntas

* Los rangos de categorías y clasificaciones se muestran de forma más intuitiva, pues se puede apreciar con más claridad y orden los límites superiores e inferiores que los delimitan, como se puede observar en la Figura 19.

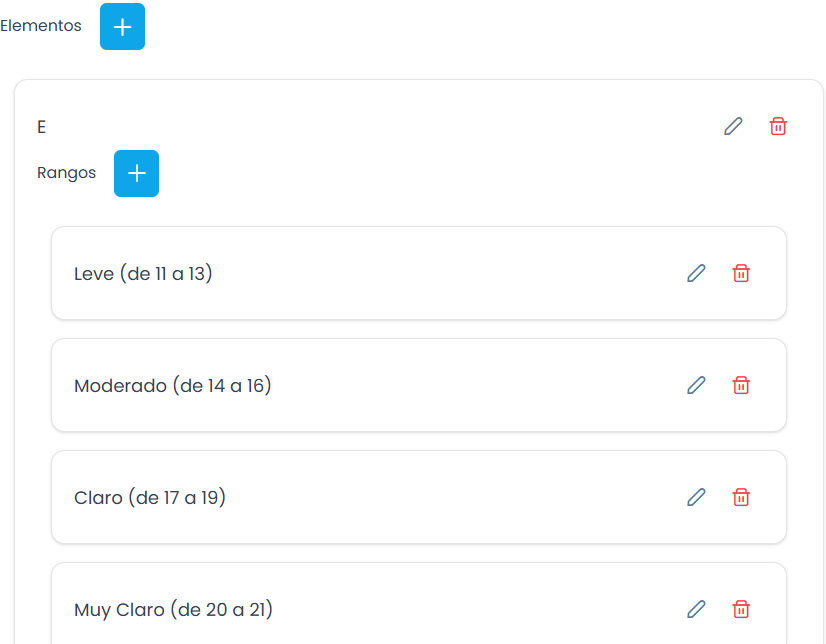


Figura 19: Fragmento de la interfaz para la creación de un test

* Se introdujeron mejoras como validaciones en tiempo real que indican las condiciones necesarias para poder avanzar en el procedimiento. Como se puede observar en la Figura 20, las condiciones están satisfechas y por tanto se puede avanzar en el formulario.

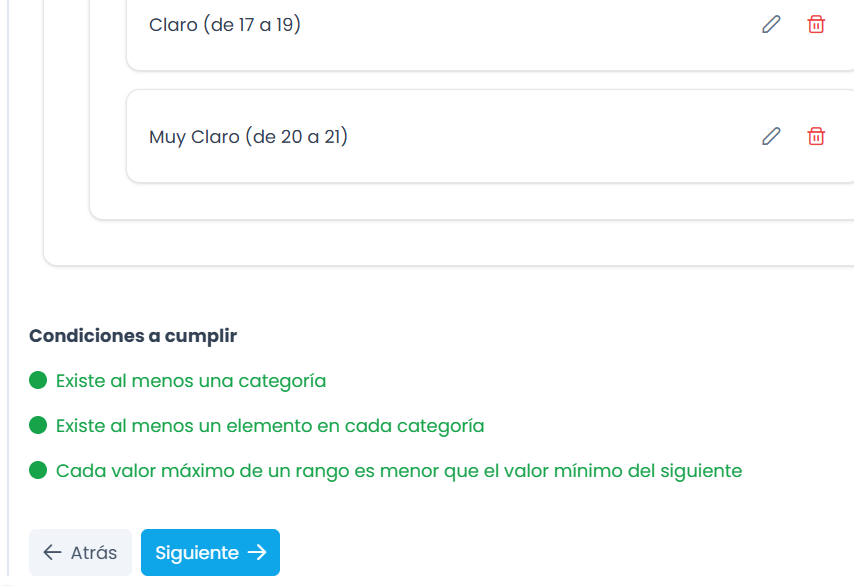


Figura 20: Fragmento de la interfaz para la creación de un test (Muestra las condiciones en tiempo real, que deben cumplirse para avanzar al siguiente paso de creación del test).

* Se introdujeron mensajes de información en los inputs del proceso de creación, para orientar al analista sobre la funcionalidad de cada parámetro a configurar.

## 2.9 Migración de la API

El *back-end* antes se encontraba conformado por una serie de controladores que se comunicaban directamente con la lógica que gestionaba las visuales. La nueva interfaz visual se comunica con el *back-end* a través de una *API* mediante peticiones *http.*

La arquitectura *back-end* sigue la estructura de controladores, encargados de definir las rutas por las que transita la información siempre y cuando las credenciales de autorización sean correctas, servicios, donde se maneja la lógica de negocio, y los modelos, que representan las entidades en base de datos para el manejo de la información. De esta forma la escalabilidad del *back-end* resulta bastante sencilla, al mantener una separación de las distintas capas.

### 2.9.1 Arquitectura del proyecto de *back-end*

El proyecto de *back-end* actual presenta la siguiente arquitectura, la cual se puede apreciar en la Figura 21 . En la carpeta raíz del proyecto se encuentran los directorios:

* ***node\_modules***: Carpeta donde se almacenan las dependencias del proyecto.
* ***src***: Carpeta donde se encuentra el código fuente del proyecto.
* ***uploads***: Almacena los archivos cargados en el sistema, dígase imágenes principalmente.

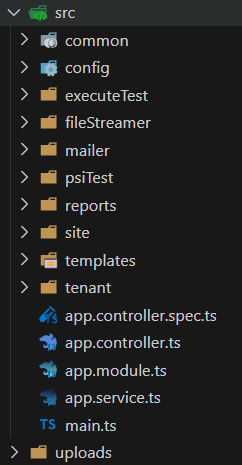


Figura 21: Jerarquía de directorios del proyecto *back-end*

El resto de archivos son de configuración, así como de información descriptiva del proyecto en sí.

En la carpeta ***src,*** el proyecto se divide en módulos dada la arquitectura que ofrece *NestJs*. Cada módulo, en caso de ser necesario, tiene una serie de carpetas que son:

* ***controllers***: agrupa los controladores del correspondiente módulo.
* ***services***: agrupa los servicios correspondientes
* ***models***: agrupa los modelos y ***subscribers*** del modelo en caso de ser necesario.
* ***dto***: donde se declaran los ***data transfer objects*** usados para la validación de la información antes de ser procesada en los controladores.

La carpeta ***config*** es la excepción a lo antes planteado. Esta carpeta contiene archivos para la configuración manual de algunos módulos proporcionados por dependencias externas, como son los casos del ***ORM*** y ***JWT***.

A continuación, se explica cada módulo de forma superficial.

* ***common***: se encuentra una serie de recursos generalizados, altamente configurables, adaptables y reutilizables para agilizar el desarrollo de los demás módulos. Son recursos que se requieren regularmente exceptuando algunos casos particulares.
* ***psiTest***: Módulo más grande, contiene la mayor cantidad de controladores, servicios y modelos, dado que es el encargado de las operaciones básicas que se realizan sobre un test: creación, eliminación, actualización y eliminación; así como de las demás entidades involucradas directamente con el test, dígase series, preguntas, las respuestas de una pregunta, las categorías y elementos a los que tributan, en cuanto tributan, entre otra serie de elementos.
* ***site***: Contiene la lógica de autenticación en el sistema.
* ***reports***: Contiene los *endpoints* para solicitar los reportes de la aplicación.
* ***site***: Contiene la lógica de autenticación en el sistema.
* ***fileStreamer***: Contiene la lógica para en envío de archivos.
* ***tenant***: Similar al módulo *psiTest,* solo que en este se trabaja con las entidades de usuarios y grupos, ya que estos son los que conforman el ***tenant*** en el sistema. Este es usado por ***site*** para la verificación de credenciales y el registro de nuevos usuarios.
* ***executeTest***: módulo muy específico, contiene la lógica de validación, almacenamiento y procesamiento de un test.

## 2.10 Despliegue de la solución

A continuación, en la Figura 22, se observa el diagrama de despliegue de la solución. En él se puede observar el nodo cliente (donde el usuario podrá interactuar con la aplicación), el servidor web (donde estará desplegada la solución), el servidor de correos, y el servidor de base de datos (donde se almacenará la base de datos del sistema). Cada nodo posee sus requisitos mínimos para funcionar.

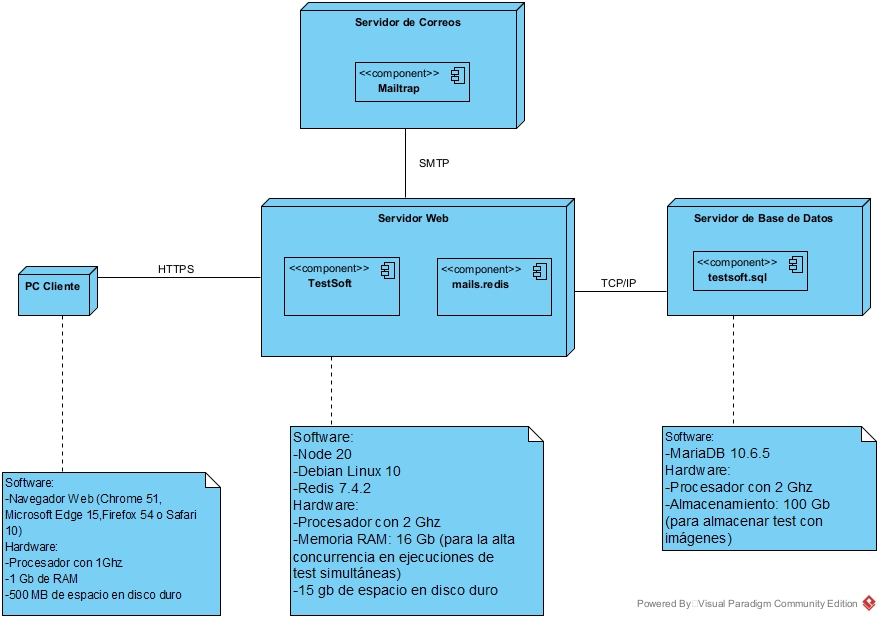


Figura 22: Diagrama de despliegue

## 2.11 Conclusiones parciales

Al concluir el capítulo se arriba a las siguientes conclusiones:

* Se presenta un diseño de base de datos modificado que permite utilizar el software como una solución SaaS.
* Los cambios en la base de datos permiten que pueda ser aplicado en cualquier entorno no solo en el académico.
* Se realizó una migración de la interfaz visual con el objetivo de mejorar la usabilidad, la experiencia de usuario, compatibilidad de navegadores y dar solución a algunos problemas como el diseño *responsive*.
* Los patrones de diseño utilizados permiten garantizar una mejor legibilidad y extensibilidad del código, en especial el patrón *Strategy*, ya que facilita en gran medida la inserción de nuevos tipos de preguntas. Este aspecto aporta a la escalabilidad de la aplicación, elemento de suma importancia en el modelo SaaS.
* Se desarrolló una API que replica el funcionamiento de la aplicación original y es capaz de comunicar la base de datos con la nueva interfaz gráfica desarrollada.
* Se presenta una arquitectura en ambos proyectos, con capacidad de extensión, escalabilidad sencilla y fácil mantenimiento.

# Capítulo 3: Validación de la solución

## 3.1 Introducción

En este capítulo se documentan las pruebas realizadas con el propósito de validar la solución propuesta. Para ello se diseñan casos de prueba que se enfocan en demostrar el funcionamiento correcto de la nueva interfaz gráfica y sus mejoras con respecto a la anterior. Además, se documenta una prueba de rendimiento realizada a la ejecución del test en ambas versiones de la herramienta.

## 3.2 Casos de prueba funcionales

Para validar el funcionamiento de la herramienta se diseñaron los siguientes casos de prueba:

* **Recibo de los test asignados a un usuario:** En este caso de prueba se pretende comprobar que los test que son pedidos a la API son los correctos según el usuario autenticado.
* **Ejecutar Test:** El presente caso de prueba se diseña con el objetivo de comprobar que la ejecución de un test determinado en la nueva interfaz devuelve el mismo resultado que en la anterior.
* **Consulta de resultados:** En este caso se pretende comprobar que la vista de muestreo de resultados, realiza su función correctamente.

### 3.2.1 Caso de Prueba: Recibo de los test asignados a un usuario

Tabla 3: Caso de Prueba: Recibo de los test asignados a un usuario

|  |  |
| --- | --- |
| **Caso de Prueba** | Recibo de los test asignados a un usuario |
| **Desarrollador** | Ernesto Alejandro Carralero Conde |
| **Probador** | Jesús Manuel Castellanos Reynaldo |
| **Objetivo de la Prueba** | Comprobar que los test recibidos en la nueva interfaz, son los asignados para el usuario autenticado |
| **Descripción de la Prueba** | Se inicia sesión en la aplicación y se dirige a la vista de selección de test |
| **Condiciones:** | |
| **Condiciones de Entrada** | **Resultados Esperados** |
| **Respuestas del usuario** | Se muestran los test correspondientes |
| **Observaciones:** | |
| Los resultados fueron parcialmente los esperados, pues, a pesar de que se mostraban los test correctos, era posible ejecutar el test MBTI, que ya se había realizado y no debería ser así, y en su lugar se debería mostrar el tiempo restante para volver a estar disponible. | |



Figura 23: Fragmento de interfaz de selección de test a ejecutar.

Una vez corregidos los errores detectados anteriormente, se volvió a ejecutar el caso de prueba, y se comprobó que el test MBTI no se puede realizar hasta que esté disponible nuevamente como se muestra en la Figura 23.

### 3.2.2 Caso de Prueba: Ejecutar Test de Belbin

Tabla 4: Caso de Prueba: Ejecutar Test de Belbin

|  |  |
| --- | --- |
| **Caso de Prueba** | Ejecutar Test de Belbin |
| **Caso de Uso** | Ejecutar Test |
| **Desarrollador** | Ernesto Alejandro Carralero Conde |
| **Probador** | Jesús Manuel Castellanos Reynaldo |
| **Objetivo de la Prueba** | Comprobar que los resultados de la ejecución del test de Belbin son correctos |
| **Descripción de la Prueba** | Se procede a ejecutar el test de Belbin, enviarlo al *back-end* para su procesamiento y esperar el resultado |
| **Condiciones:** | |
| **Condiciones de Entrada** | **Resultados Esperados** |
| **Respuestas del usuario** | Los resultados mostrados en la nueva interfaz, deben ser los mismos que en la antigua interfaz al ejecutar el test y responderlo de la misma forma |
| **Observaciones:** | |
| Los resultados fueron satisfactorios. | |

Como se observa en la Figura 24, los resultados son los mismos en las dos interfaces.

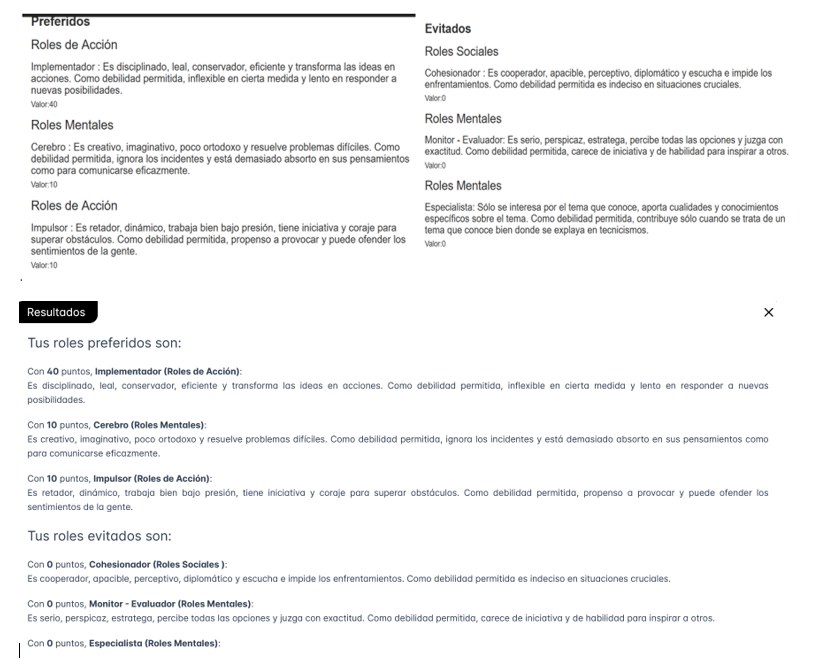


Figura 24: a) Fragmento de la interfaz de visualización de resultados del test de Belbin en la interfaz antigua (sección superior). b) Fragmento de la interfaz de visualización de resultados del test de Belbin en la nueva interfaz (sección inferior).

### 3.2.3 Caso de Prueba: Ejecutar Test MBTI

Tabla 5: Caso de Prueba: Ejecutar Test MBTI

|  |  |
| --- | --- |
| **Caso de Prueba** | Ejecutar Test MBTI |
| **Caso de Uso** | Ejecutar Test |
| **Desarrollador** | Jesús Manuel Castellanos Reynaldo |
| **Probador** | Ernesto Alejandro Carralero Conde |
| **Objetivo de la Prueba** | Comprobar que los resultados de la ejecución del test MBTI son correctos |
| **Descripción de la Prueba** | Se procede a ejecutar el test MBTI, enviarlo al *back-end* para su procesamiento y esperar el resultado |
| **Condiciones:** | |
| **Condiciones de Entrada** | **Resultados Esperados** |
| **Respuestas del usuario** | Los resultados mostrados en la nueva interfaz, deben ser los mismos que en la antigua interfaz al ejecutar el test y responderlo de la misma forma |
| **Observaciones:** | |
| Los valores de los elementos no se muestran de forma correcta. | |

El error estaba relacionado con la utilización en el *front-end* de un atributo inexistente en la respuesta recibida desde el *back-end.* Se volvió a ejecutar el caso de prueba una vez corregido el error, y se comprobó (como se muestra en la Figura 23), que, tanto en la interfaz anterior como en la nueva, se muestran los mismos valores en las respuestas.

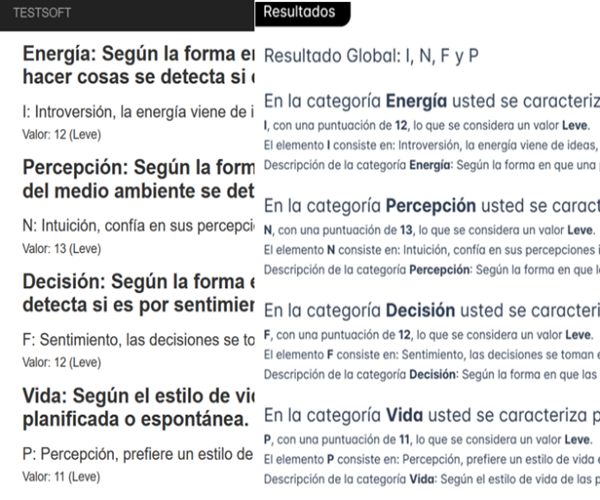


Figura 25: a) Fragmento de la interfaz de visualización de resultados del test de MBTI en la interfaz antigua (sección izquierda). b) Fragmento de la interfaz de visualización de resultados del test de MBTI en la nueva interfaz (sección derecha).

3.2.4 Caso de Prueba: Consulta de resultados de test MBTI

Tabla 6: Caso de Prueba: Consulta de resultados de test MBTI

|  |  |
| --- | --- |
| **Caso de Prueba** | Consulta de resultados |
| **Caso de Uso** | Consulta de resultados de test MBTI |
| **Desarrollador** | Jesús Manuel Castellanos Reynaldo |
| **Probador** | Ernesto Alejandro Carralero Conde |
| **Objetivo de la Prueba** | Comprobar que se muestra los resultados correctamente |
| **Descripción de la Prueba** | Se autentica en la aplicación y se dirige a la vista de muestreo de resultados |
| **Condiciones:** | |
| **Condiciones de Entrada** | **Resultados Esperados** |
| **Respuestas del usuario** | Los resultados mostrados en la interfaz son los correspondientes a la previa ejecución de los test por parte de ese usuario |
| **Observaciones:** | |
| Las fechas no se mostraron de forma correcta. | |

El error estaba relacionado con la conversión de las fechas recibidas de formato JSON a un objeto *Date* de *javascript*. Se volvió a ejecutar el caso de prueba una vez corregido el error, y se comprobó (como se muestra en la Figura 26) que las fechas se muestran de forma correcta.

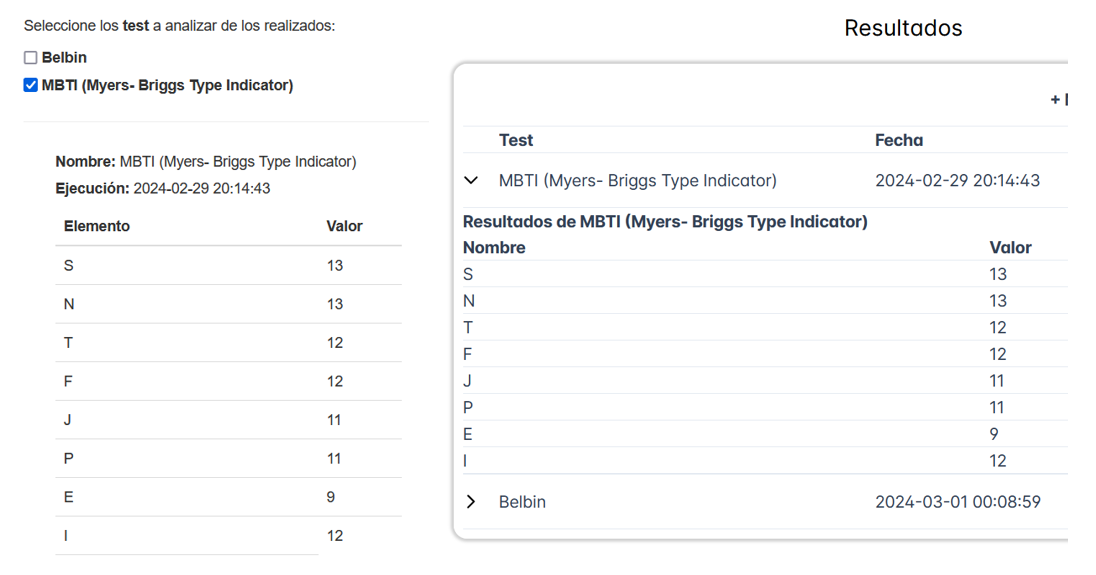


Figura 26: a) Fragmento de la interfaz de visualización de resultados de test en la interfaz antigua (sección izquierda). b) Fragmento de la interfaz de visualización de resultados del test en la nueva interfaz (sección derecha).

### 3.2.5 Caso de Prueba: Creación de copia del test Belbin

Tabla 7: Caso de Prueba: Creación de copia del test Belbin

|  |  |
| --- | --- |
| **Caso de Prueba** | Creación de test |
| **Caso de Uso** | Creación de copia del test de Belbin |
| **Desarrollador** | Jesús Manuel Castellanos Reynaldo  Ernesto Alejandro Carralero Conde |
| **Probador** | Ernesto Alejandro Carralero Conde |
| **Objetivo de la Prueba** | Comprobar que se pueden crear test psicológicos correctamente en la nueva versión de la aplicación |
| **Descripción de la Prueba** | Se autentica en la aplicación y se dirige a la vista de gestión de test psicológicos, donde se procede a crear una copia del test de Belbin, con los mismos parámetros que ya posee el existente en la aplicación. Luego, se procede a ejecutar ambas versiones para comprobar que devuelvan los mismos resultados. |
| **Condiciones:** | |
| **Condiciones de Entrada** | **Resultados Esperados** |
| **Respuestas del usuario** | Los resultados mostrados en ambas versiones del test de Belbin son exactamente los mismos. |
| **Observaciones:** | |
|  | |

## 

TestSoft, como aplicación web, requiere técnicas de pruebas de caja negra para garantizar su funcionalidad, usabilidad y calidad desde la perspectiva del usuario. Estas pruebas se centran en evaluar el comportamiento externo de la aplicación sin necesidad de conocer su código interno, lo que permite verificar que las funcionalidades clave, como la creación, aplicación y procesamiento de test psicológicos, funcionen correctamente.

Las clases de equivalencia son una técnica de pruebas de caja negra que divide los datos de entrada en grupos que se espera que se comporten de manera similar. Esta técnica es especialmente útil para validar la ejecución de un test psicológico, ya que reduce el número de casos de prueba necesarios al seleccionar valores representativos de cada clase.

En el contexto de la validación y pruebas de una aplicación como TestSoft, el uso de Playwright y Postman es altamente beneficioso debido a sus capacidades complementarias. Playwright es una herramienta de automatización de pruebas que permite simular interacciones de usuario en una interfaz web, lo que es ideal para probar la ejecución de test psicológicos desde la perspectiva del usuario final. Con Playwright, se pueden automatizar escenarios como el envío de respuestas, la navegación entre preguntas y la validación de resultados, asegurando que la interfaz de usuario funcione correctamente en diferentes navegadores y dispositivos. Además, su capacidad para manejar pruebas de extremo a extremo permite verificar que todos los componentes de la aplicación trabajen en conjunto de manera fluida, lo que es esencial para garantizar una experiencia de usuario consistente y sin errores.

Por otro lado, Postman es una herramienta especializada en pruebas de API, lo que resulta invaluable para validar el backend de TestSoft. Con Postman, se pueden realizar pruebas exhaustivas de los endpoints que gestionan la creación, aplicación y procesamiento de los test psicológicos. Esto incluye verificar que las respuestas se almacenen correctamente, que los cálculos de los resultados sean precisos y que el sistema maneje adecuadamente casos extremos, como el envío de un cuerpo vacío o respuestas inválidas. La capacidad de Postman para automatizar pruebas y generar informes detallados facilita la identificación rápida de errores y la validación de la lógica del servidor. Juntas, Playwright y Postman proporcionan una cobertura completa de pruebas, tanto en el frontend como en el backend, asegurando que TestSoft funcione de manera robusta y confiable en todos sus aspectos.

**Diseño de los casos de pruebas y resultados alcanzados**

Esta sección tiene como propósito mostrar el diseño de casos de pruebas y los resultados alcanzados con cada uno de ellos. Para la validación de la sección más importante de la aplicación, o sea, la ejecución de test psicológicos, se crearon casos de prueba donde se evalúa el proceso de ejecutar los test de Belbin y MBTI.

Con el objetivo de comprobar el correcto funcionamiento de algunas de las

principales funcionalidades del sistema se diseñaron y ejecuta  casos de

prueba asociados al caso de uso del sistema “Ejecutar Test”, donde para comprobar que los test creados se ejecutan correctamente, o sea, se aplican, se procesan y se visualizan los resultados como está previsto en el test. Los casos de prueba se dividen en front-end y back-end, realizados con las herramientas Playwright y Postman respectivamente. Las comprobaciones realizadas para validar los resultados son posibles gracias a la ejecución de estos test en unos archivos Excel entregados en la asignatura de “Gestión de Proyectos".

**Caso de Prueba: Ejecutar Test de Belbin (front-end)**

En la Tabla 1 se puede observar el caso de prueba “ Ejecutar Test de Belbin (front-end)“. En este caso de prueba se utilizan 3 clases de equivalencia:

* Clase válida: Enviar respuestas correctas del test psicológico (1er flujo).
* Clase inválida: Enviar un cuerpo vacío (2do flujo).
* Clase inválida: Enviar respuestas antes de que se cumpla el tiempo en que vuelve a estar disponible el test (3er flujo).

Tabla 1: Caso de Prueba: Ejecutar Test de Belbin (front-end)

|  |  |
| --- | --- |
| **Caso de Prueba** | Ejecutar Test de Belbin (front-end) |
| **Caso de Uso** | Ejecutar Test |
| **Desarrollador** | Jesús Manuel Castellanos Reynaldo |
| **Probador** | Ernesto Alejandro Carralero Conde  (Playwright) |
| **Objetivo de la Prueba** | Comprobar que los resultados de la ejecución del test de Belbin son correctos |
| **Descripción de la Prueba** | Se procede a ejecutar el test de Belbin, enviar la respuesta para su procesamiento y esperar el resultado |
|  |  |
| **1-Condición de ejecución** | Verificar una ejecución correcta del test (clase válida) |
| **Condiciones previas** | Usuario autenticado como ejecutor de test. |
| **Entradas de prueba** | Se procede a insertar las mismas respuestas que en el test realizado en el Excel. |
| **Puntos de Observación** | El test no devolvió los resultados adecuados |
| **Puntos de control** |  |
| **Resultados esperados** | Se muestra una pantalla con los resultados de la ejecución del test, y estos deben ser iguales a los resultados que ofrece el Excel. |
| **Condiciones Posteriores** |  |
|  |  |
| **2-Condición de ejecución** | Verificar una ejecución incompleta del test (clase inválida) |
| **Condiciones previas** | Usuario autenticado como ejecutor de test. |
| **Entradas de prueba** | Se procede a guardar el test sin respuestas. |
| **Puntos de Observación** | Los resultados reales son los esperados |
| **Puntos de control** |  |
| **Resultados esperados** | Se muestran mensajes de error que ofrecen información al usuario acerca de las preguntas que faltan por responder. |
| **Condiciones Posteriores** |  |
|  |  |
| **3-Condición de ejecución** | Verificar una ejecución antes de volver a estar disponible nuevamente un test (clase inválida) |
| **Condiciones previas** | Usuario autenticado como ejecutor de test.  Poseer una ejecución de test de Belbin con menos de 4 años de diferencia de la fecha actual. |
| **Entradas de prueba** | Se procede a intentar ejecutar un test de Belbin. |
| **Puntos de Observación** | Botones de scroll innecesarios en los test a seleccionar |
| **Puntos de control** |  |
| **Resultados esperados** | Se bloquea el acceso al test, y se muestra el tiempo real que resta para volver a estar disponible |
| **Condiciones Posteriores** |  |

Como se puede observar en la tabla anterior, para el primer flujo no se obtuvieron los resultados esperados, ya que la aplicación no devolvió los resultados del test indicados. Se detectó un error en el orden en que se retornaban los resultados, el cual fue corregido. La prueba fue llevada a cabo nuevamente, y los resultados fueron satisfactorios. Para el segundo escenario los resultados fueron los esperados, igual que para el tercero, solo con la particularidad de que se detectaron  barras de scroll innecesarias en los componentes de selección del test a ejecutar.

**Caso de Prueba: Ejecutar Test de Belbin (back-end)**

En la Tabla 2 se puede observar el caso de prueba “ Ejecutar Test de Belbin (back-end)“. En este caso de prueba se utilizan 3 clases de equivalencia:

* Clase válida: Enviar respuestas correctas del test psicológico (1er flujo).
* Clase inválida: Enviar un cuerpo vacío (2do flujo).
* Clase inválida: Enviar respuestas antes de que se cumpla el tiempo en que vuelve a estar disponible el test (3er flujo).

Tabla 2: Caso de Prueba: Ejecutar Test de Belbin (back-end)

|  |  |
| --- | --- |
| **Caso de Prueba** | Ejecutar Test de Belbin (back-end) |
| **Caso de Uso** | Ejecutar Test |
| **Desarrollador** | Ernesto Alejandro Carralero Conde |
| **Probador** | Jesús Manuel Castellanos Reynaldo  (Postman) |
| **Objetivo de la Prueba** | Comprobar que los resultados de la ejecución del test de Belbin son correctos |
| **Descripción de la Prueba** | Se procede a enviar una ejecución de test de Belbin y esperar el resultado |
|  |  |
| **1-Condición de ejecución** | Verificar una ejecución correcta del test (clase válida) |
| **Condiciones previas** | Usuario autenticado como ejecutor de test. |
| **Entradas de prueba** | Se procede a insertar las mismas respuestas que en el test realizado en el Excel. |
| **Puntos de Observación** | El test no devolvió los resultados adecuados |
| **Puntos de control** |  |
| **Resultados esperados** | Los resultados de la ejecución del test deben ser iguales a los resultados que ofrece el Excel. |
| **Condiciones Posteriores** |  |
|  |  |
| **2-Condición de ejecución** | Verificar una ejecución incompleta del test (clase inválida) |
| **Condiciones previas** | Usuario autenticado como ejecutor de test. |
| **Entradas de prueba** | Se procede a guardar el test sin respuestas. |
| **Puntos de Observación** | Al realizar la prueba directamente a la API, devuelve un código de respuesta 201 y guarda una ejecución de test, cuando no dberia hacerlo. |
| **Puntos de control** |  |
| **Resultados esperados** | Código de respuesta 400 |
| **Condiciones Posteriores** |  |
|  |  |
| **3-Condición de ejecución** | Verificar una ejecución antes de volver a estar disponible nuevamente un test (clase inválida) |
| **Condiciones previas** | Usuario autenticado como ejecutor de test.  Poseer una ejecución de test de Belbin con menos de 4 años de diferencia de la fecha actual. |
| **Entradas de prueba** | Se procede a intentar ejecutar un test de Belbin. |
| **Puntos de Observación** | Es posible guardar una nueva ejecución del test, lo que es incorrecto ya que se debería de solo aceptar ejecuciones de una fecha posterior a los 4 años desde la última ejecución |
| **Puntos de control** |  |
| **Resultados esperados** | Código de respuesta 400 |
| **Condiciones Posteriores** |  |

Como se puede observar en la tabla anterior y en la Figura 1, para el primer flujo no se obtuvieron los resultados esperados, ya que la aplicación no devolvió los resultados del test indicados. Se detectó un error en el orden en que se retornaban los resultados, el cual fue corregido. La prueba fue llevada a cabo nuevamente, y los resultados fueron satisfactorios. Para el segundo y tercer escenario, los resultados tampoco fueron los esperados, pues la aplicación debería haber enviado un código de error 400, indicando un error en los datos enviados, como se puede observar en la Figura 2. Por lo tanto, se concluye que es necesario implementar estas validaciones en el lado del servidor, ya que actualmente solo existen en el lado del cliente.

Figura 1: Resultado fallido de la ejecución en Postman del Caso de Prueba: Ejecutar Test de Belbin (back-end) (Escenario 1) .

Figura 2: Resultado fallido de la ejecución en Postman del Caso de Prueba: Ejecutar Test de Belbin (back-end) (Escenario 2) .

**Caso de Prueba: Ejecutar Test de MBTI (front-end)**

En la Tabla 3 se puede observar el caso de prueba “ Ejecutar Test de MBTI (front-end)“. En este caso de prueba se utilizan 3 clases de equivalencia:

* Clase válida: Enviar respuestas correctas del test psicológico (1er flujo).
* Clase inválida: Enviar un cuerpo vacío (2do flujo).
* Clase inválida: Enviar respuestas antes de que se cumpla el tiempo en que vuelve a estar disponible el test (3er flujo).

Tabla 3: Caso de Prueba: Ejecutar Test de MBTI (front-end)

|  |  |
| --- | --- |
| **Caso de Prueba** | Ejecutar Test de MBTI (front-end) |
| **Caso de Uso** | Ejecutar Test |
| **Desarrollador** | Jesús Manuel Castellanos Reynaldo |
| **Probador** | Ernesto Alejandro Carralero Conde  (Playwright) |
| **Objetivo de la Prueba** | Comprobar que los resultados de la ejecución del test de MBTI son correctos |
| **Descripción de la Prueba** | Se procede a ejecutar el test de MBTI, enviar la respuesta para su procesamiento y esperar el resultado |
|  |  |
| **1-Condición de ejecución** | Verificar una ejecución correcta del test (clase válida) |
| **Condiciones previas** | Usuario autenticado como ejecutor de test. |
| **Entradas de prueba** | Se procede a insertar las mismas respuestas que en el test realizado en el Excel. |
| **Puntos de Observación** | El test no devolvió los resultados adecuados |
| **Puntos de control** |  |
| **Resultados esperados** | Se muestra una pantalla con los resultados de la ejecución del test, y estos deben ser iguales a los resultados que ofrece el Excel. |
| **Condiciones Posteriores** |  |
|  |  |
| **2-Condición de ejecución** | Verificar una ejecución incompleta del test (clase inválida) |
| **Condiciones previas** | Usuario autenticado como ejecutor de test. |
| **Entradas de prueba** | Se procede a guardar el test sin respuestas. |
| **Puntos de Observación** | Los resultados reales son los esperados |
| **Puntos de control** |  |
| **Resultados esperados** | Se muestran mensajes de error que ofrecen información al usuario acerca de las preguntas que faltan por responder. |
| **Condiciones Posteriores** |  |
|  |  |
| **3-Condición de ejecución** | Verificar una ejecución antes de volver a estar disponible nuevamente un test (clase inválida) |
| **Condiciones previas** | Usuario autenticado como ejecutor de test.  Poseer una ejecución de test de Belbin con menos de 2 años de diferencia de la fecha actual. |
| **Entradas de prueba** | Se procede a intentar ejecutar un test de MBTI. |
| **Puntos de Observación** | Botones de scroll innecesarios en los test a seleccionar |
| **Puntos de control** |  |
| **Resultados esperados** | Se bloquea el acceso al test, y se muestra el tiempo real que resta para volver a estar disponible |
| **Condiciones Posteriores** |  |

Como se puede observar en la tabla anterior, para el primer flujo no se obtuvieron los resultados esperados, ya que la aplicación no devolvió los resultados del test indicados. Se detectó un error en el orden en que se retornaban los resultados, el cual fue corregido. La prueba fue llevada a cabo nuevamente, y los resultados fueron satisfactorios. Para el segundo escenario los resultados fueron los esperados, igual que para el tercero, solo con la particularidad de que se detectaron  barras de scroll innecesarias en los componentes de selección del test a ejecutar.

**Caso de Prueba: Ejecutar Test de MBTI (back-end)**

En la Tabla 4 se puede observar el caso de prueba “ Ejecutar Test de MBTI (back-end)“. En este caso de prueba se utilizan 3 clases de equivalencia:

* Clase válida: Enviar respuestas correctas del test psicológico (1er flujo).
* Clase inválida: Enviar un cuerpo vacío (2do flujo).
* Clase inválida: Enviar respuestas antes de que se cumpla el tiempo en que vuelve a estar disponible el test (3er flujo).

Tabla 4: Caso de Prueba: Ejecutar Test de MBTI (back-end)

|  |  |
| --- | --- |
| **Caso de Prueba** | Ejecutar Test de MBTI (back-end) |
| **Caso de Uso** | Ejecutar Test |
| **Desarrollador** | Ernesto Alejandro Carralero Conde |
| **Probador** | Jesús Manuel Castellanos Reynaldo  (Postman) |
| **Objetivo de la Prueba** | Comprobar que los resultados de la ejecución del test de MBTI son correctos |
| **Descripción de la Prueba** | Se procede a enviar una ejecución de test de MBTI y esperar el resultado |
|  |  |
| **1-Condición de ejecución** | Verificar una ejecución correcta del test (clase válida) |
| **Condiciones previas** | Usuario autenticado como ejecutor de test. |
| **Entradas de prueba** | Se procede a insertar las mismas respuestas que en el test realizado en el Excel. |
| **Puntos de Observación** | El test no devolvió los resultados adecuados |
| **Puntos de control** |  |
| **Resultados esperados** | Los resultados de la ejecución del test deben ser iguales a los resultados que ofrece el Excel. |
| **Condiciones Posteriores** |  |
|  |  |
| **2-Condición de ejecución** | Verificar una ejecución incompleta del test (clase inválida) |
| **Condiciones previas** | Usuario autenticado como ejecutor de test. |
| **Entradas de prueba** | Se procede a guardar el test sin respuestas. |
| **Puntos de Observación** | Al realizar la prueba directamente a la API, devuelve un código de respuesta 201 y guarda una ejecución de test, cuando no debería hacerlo. |
| **Puntos de control** |  |
| **Resultados esperados** | Código de respuesta 400 |
| **Condiciones Posteriores** |  |
|  |  |
| **3-Condición de ejecución** | Verificar una ejecución antes de volver a estar disponible nuevamente un test (clase inválida) |
| **Condiciones previas** | Usuario autenticado como ejecutor de test.  Poseer una ejecución de test de Belbin con menos de 2 años de diferencia de la fecha actual. |
| **Entradas de prueba** | Se procede a intentar ejecutar un test de MBTI. |
| **Puntos de Observación** | Es posible guardar una nueva ejecución del test, lo que es incorrecto ya que se debería de solo aceptar ejecuciones de una fecha posterior a los 2 años desde la última ejecución |
| **Puntos de control** |  |
| **Resultados esperados** | Código de respuesta 400 |
| **Condiciones Posteriores** |  |

Como se puede observar en la tabla anterior y en la Figura 3, para el primer flujo no se obtuvieron los resultados esperados, ya que la aplicación no devolvió los resultados del test indicados. Se detectó un error en el orden en que se retornaban los resultados, el cual fue corregido. La prueba fue llevada a cabo nuevamente, y los resultados fueron satisfactorios. Para el segundo y tercer escenario, los resultados tampoco fueron los esperados, pues la aplicación debería haber enviado un código de error 400, indicando un error en los datos enviados, como se puede observar en a Figura 4. Por lo tanto, se concluye que es necesario implementar estas validaciones en el lado del servidor, ya que actualmente solo existen en el lado del cliente.

Figura 3: Resultado fallido de la ejecución en Postman del Caso de Prueba: Ejecutar Test de MBTI (back-end) (Escenario 1) .

Figura 4: Resultado de la ejecución en Postman del Caso de Prueba: Ejecutar Test de MBTI (back-end) (Escenario 2) .

De forma general, en las pruebas de front-end se detectaron problemas en la devolución de los resultados de la ejecución de test psicológicos tanto para Belbin como para MBTI, como se puede observar en la Figura 5. Estos problemas fueron corregidos, y las pruebas se ejecutaron nuevamente, como se puede observar en la Figura 6, para validar los arreglos realizados, como se había mencionado con anterioridad.

Figura 5: Resultados fallidos de la ejecución en Playwright de las Pruebas Diseñadas.

Figura 6: Resultados de la ejecución en Playwright de las Pruebas Diseñadas, luego de la corrección de errores.

**Muestras del uso de las herramientas**

A continuación se podrán observar capturas del uso de las herramientas para la realización de las pruebas diseñadas.

**Postman**

En esta sección,  se pueden observar las distintas pruebas implementadas en Postman.

Figura 7: Implementación de Prueba en Postman para validar el Caso de Prueba: Ejecutar Test de Belbin (back-end) (1).

Figura 8: Implementación de Prueba en Postman para validar el Caso de Prueba: Ejecutar Test de Belbin (back-end) (Escenario 2).

Figura 9: Implementación de Prueba en Postman para validar el Caso de Prueba: Ejecutar Test de MBTI (back-end) (Escenario 1).

Figura 10: Implementación de Prueba en Postman para validar el Caso de Prueba: Ejecutar Test de MBTI (back-end) (Escenario 2).

**Playwright**

En esta sección,  se pueden observar las distintas pruebas implementadas en Playwright.

Figura 11: Implementación de Prueba en Postman para validar el Caso de Prueba: Ejecutar Test de Belbin (front-end) (Escenario 1).

Figura 12: Implementación de Prueba en Postman para validar el Caso de Prueba: Ejecutar Test de Belbin (front-end) (Escenario 2).

Figura 13: Implementación de Prueba en Postman para validar el Caso de Prueba: Ejecutar Test de Belbin (front-end) (Escenario 3).

Figura 14: Implementación de Prueba en Postman para validar el Caso de Prueba: Ejecutar Test de MBTI (front-end) (Escenario 1).

Figura 15: Implementación de Prueba en Postman para validar el Caso de Prueba: Ejecutar Test de MBTI (front-end) (Escenario 2).

Figura 16: Implementación de Prueba en Postman para validar el Caso de Prueba: Ejecutar Test de MBTI (front-end) (Escenario 3).

**Conclusiones**

Con la realización de este trabajo se arribó a las siguientes conclusiones:

* Se identificaron defectos de validación en el backend que no estaban presentes en el frontend, lo que resalta la importancia de implementar validaciones consistentes en ambos lados de la aplicación. Esto asegura que los datos sean correctamente procesados y que se eviten inconsistencias o errores que puedan afectar la integridad del sistema.
* Las pruebas de caja negra, como las particiones de equivalencia, demostraron ser efectivas para detectar errores en el manejo de entradas válidas e inválidas. Estas técnicas permitieron identificar casos críticos, como el envío de respuestas vacías o fuera de rango, que no estaban siendo manejados adecuadamente en el backend.
* La integración de herramientas como Playwright y Postman fue fundamental para realizar pruebas exhaustivas tanto en el frontend como en el backend. Playwright permitió validar la interfaz de usuario y la experiencia del usuario, mientras que Postman facilitó la verificación de la lógica del servidor y el manejo de respuestas incorrectas.
* Se recomienda implementar validaciones redundantes en ambos lados (frontend y backend) para garantizar un manejo robusto de los datos. Esto incluye validar formatos, rangos y tipos de datos, así como manejar adecuadamente casos extremos y entradas inesperadas.
* La detección temprana de defectos a través de pruebas automatizadas reduce costos y tiempo en el ciclo de desarrollo. Este trabajo demostró que las pruebas continuas y bien diseñadas son esenciales para mantener la calidad del software y evitar problemas en producción.

## 3.3 Pruebas de rendimiento

Para demostrar lo investigado acerca de las ventajas en rendimiento de *NestJs* con respecto a Yii2, se desarrolló la funcionalidad de ejecución del test como primer paso. Luego, se ejecutaron peticiones de ejecución de test de MBTI con las mismas respuestas a ambas versiones, para comparar los resultados. Para ello se usó el software *Postman*, el cual permite mostrar el tiempo que se demora una API en procesar cada petición y devolver una respuesta. En la Tabla 8 se puede observar el tiempo de 10 peticiones y su tiempo promedio.

Tabla 8: Resultados de pruebas de rendimiento realizadas a ambas versiones de la aplicación

|  |  |  |
| --- | --- | --- |
| **Petición** | **Tiempo de respuesta de la versión en *Yii2* (ms)** | **Tiempo de respuesta de la versión en *NestJs* (ms)** |
| 1 | 547 | 251 |
| 2 | 498 | 289 |
| 3 | 488 | 217 |
| 4 | 491 | 179 |
| 5 | 513 | 160 |
| 6 | 482 | 167 |
| 7 | 675 | 179 |
| 8 | 472 | 153 |
| 9 | 478 | 160 |
| 10 | 459 | 145 |
| **Promedio** | 510.3 | **190** |

Como se puede observar en la Tabla 8, la nueva versión posee un tiempo de latencia en promedio mucho menor que el tiempo de la versión antigua, lo que se traduce en que la nueva versión trabaja aproximadamente 2.68 veces más rápido, resultado que favorece la decisión tomada.

## 3.4 Conclusiones parciales

Las pruebas realizadas en el capítulo permiten arribar a las siguientes conclusiones:

* La nueva interfaz gráfica se conecta correctamente con la API, y esta a su vez, con la base de datos de la aplicación.
* Los test implementados devuelven resultados correctos e indican las preguntas con respuestas incorrectas satisfactoriamente.
* La nueva interfaz presenta mejorías notables de usabilidad, lo que hacen de TestSoft un proyecto más llamativo a nuevos clientes.
* La nueva API ofrece alrededor de 2.68 veces la velocidad de la anterior, lo que constituye una mejora considerable.
* Los errores surgidos durante la fase de testeo, fueron solucionados.

# Conclusiones

Las principales conclusiones a las que se llega, dado los resultados de este trabajo, son las siguientes:

* Se propuso un diseño de base de datos modificado que permitió la migración la herramienta TestSoft hacia una solución SaaS.
* La nueva interfaz gráfica presenta una gran mejoría en la experiencia de usuario y usabilidad de la herramienta.
* Dada la arquitectura establecida en ambos en el proyecto*,* la escalabilidad y mantenimiento del software es mucho más sencilla debido a la separación modular de sus funcionalidades.
* Se comprobó que la herramienta funciona correctamente pues la conexión entre la interfaz gráfica y la API es satisfactoria y los test psicológicos incluidos en el proceso ofrecen resultados correctos.
* Se obtuvo una versión más rápida de la herramienta que favorece los requisitos de alta concurrencia que puede solicitar un sistema SaaS.

# Recomendaciones

Las recomendaciones que se derivan de este trabajo son las siguientes:

* Incluir nuevos tipos de preguntas para ampliar el alcance de la herramienta a nuevos test psicológicos.
* Analizar la posibilidad de encriptar los datos relacionados a las ejecuciones de test, para aumentar la privacidad de los usuarios analizados.
* Añadir la posibilidad de visualizar las respuestas de una ejecución de test determinada, desde un analista autorizado. Esto puede enriquecer el análisis de los resultados y podría ayudar a sacar conclusiones sobre el rendimiento y la toma de decisiones del usuario examinado.
* Explorar la oportunidad de añadir nuevos parámetros configurables para aumentar el rango de test psicológicos que se puedan incluir en la herramienta.

# Referencias bibliográficas

[1] A. Goswami, "Human resource management and its importance for today’s organizations," *Journal of Advances and Scholarly Researches in Allied Education,* vol. 15, no. 3, pp. 128-135, 2018.

[2] J. B. Barney and P. M. Wright, "On becoming a strategic partner: The role of human resources in gaining competitive advantage," *Human Resource Management: Published in Cooperation with the School of Business Administration, The University of Michigan and in alliance with the Society of Human Resources Management,* vol. 37, no. 1, pp. 31-46, 1998.

[3] T. W. Schultz, *Investment in human capital. The role of education and of research*. New York: NY: Free Press, 1971, p. 272.

[4] F. Lotito, "Test Psicológicos y Entrevistas: Usos y Aplicaciones Claves En El Proceso De Selección E Integración De Personas a Las Empresas (Psychological Tests and Interviews: Key Uses and Applications in the Selection and Integration of Companies)," *RAN-Revista Academia & Negocios,* vol. 1, no. 2, 2015.

[5] S. J. Stabile, "The use of personality tests as a hiring tool: is the benefit worth the cost?," *U. Pa. J. Lab. & Emp. L.,* vol. 4, p. 279, 2001.

[6] R. Dana, E. Demeter, S. Ignat, and G. Rad, "Digitization of Everything, the world of 0s and 1s, emerging trends in psychological assessment," *Agora Psycho-Pragmatica,* vol. 14, no. 1, 2020.

[7] T. Ostermann, J. P. Röer, and M. J. Tomasik, "Digitalization in psychology: A bit of challenge and a byte of success," *Patterns,* vol. 2, no. 10, 2021.

[8] R. Arslan, "A review on ethical issues and rules in psychological assessment," *Journal of Family Counseling and Education,* vol. 3, no. 1, pp. 17-29, 2018.

[9] P. Succes. (2024, 20 de octubre). *Psychometric Success - Free Practice Aptitude Tests*. Available: <http://www.psychometric-success.com/>

[10] O. S. P. Project. (2019, 20 de octubre). *Open Source Psychometrics Project*. Available: <http://openpsychometrics.org/>

[11] S. T. Mueller. (2019, 21 de octubre). *PEBL: The Psychology Experiment Building Language*. Available: <https://pebl.sourceforge.net/>

[12] H. Inc. (2018, 21 de octubre). *HRPersonality*. Available: <https://www.hrpersonality.com/>

[13] T. Ediciones. (2024, 21 de octubre). *e-TEAediciones: El sistema para la aplicación y corrección online de tests.* . Available: <http://www.teaediciones.com/landing/servicios/e-teaediciones/index-etea.html#eTEA>

[14] P. A. T. Ltd. (2024, 21 de octubre). *Practice Aptitude Tests for Job Applicants & Graduates*. Available: <https://www.practiceaptitudetests.com/>

[15] C. INC. (2024, 21 de octubre). *MBTIonline.com*. Available: <https://www.mbtionline.com/AbouttheMBTI>

[16] T. G. Alvarez, "Herramienta para la creación, aplicación y análisis de los resultados de test psicológicos utilizados como apoyo en los procesos de gestión del capital humano," Trabajo para optar por el título de Ingeniería Informática, Facultad de Ingeniería Informática, Universidad Tecnológica de La Habana “José Antonio Echeverría”, La Habana, Cuba, 2018.

[17] D. L. González, "Implementación de mejoras a la herramienta TestSoft para facilitar la incorporación, creación, aplicación, procesamiento y análisis de test psicológicos," Trabajo para optar al título de Ingeniero Informático, Facultad de Ingeniería Informática, Universidad Tecnológica de La Habana “José Antonio Echeverría”, La Habana, 2021.

[18] T. G. Alvarez, "Entrevista con el proveedor de TestSoft," J. M. C. Reynaldo and E. A. Carralero, Eds., ed. La Habana, Cuba, 2024.

[19] G. Kulkarni, J. Gambhir, and R. Palwe, "Cloud computing-software as service," *International Journal of Cloud Computing and Services Science,* vol. 1, no. 1, pp. 11-16, 2012.

[20] L. Qian, Z. Luo, Y. Du, and L. Guo, "Cloud computing: An overview," in *Cloud Computing: First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings 1*, 2009, pp. 626-631: Springer.

[21] Y. Wei and M. B. Blake, "Service-oriented computing and cloud computing: Challenges and opportunities," *IEEE Internet Computing,* vol. 14, no. 6, pp. 72-75, 2010.

[22] S. Satyanarayana, "Cloud computing: SAAS," *Computer Sciences and Telecommunications,* no. 4, pp. 76-79, 2012.

[23] F. Aslam, "The Benefits and Challenges of Customization within SaaS Cloud Solutions," *American Journal of Data, Information and Knowledge Management,* vol. 4, pp. 14-22, 07/28 2023.

[24] V. H. Pinto, H. J. Luz, R. R. Oliveira, P. S. Souza, and S. R. Souza, "A Systematic Mapping Study on the Multi-tenant Architecture of SaaS Systems," in *SEKE*, 2016, pp. 396-401.

[25] F. Chong, G. Carraro, and R. Wolter, "Multi-tenant data architecture," *MSDN Library, Microsoft Corporation,* pp. 14-30, 2006.

[26] P. Mangwani, N. Mangwani, and S. Motwani, "Evaluation of a Multitenant SaaS Using Monolithic and Microservice Architectures," *SN Computer Science,* vol. 4, 01/31 2023.

[27] M. Bauer, "Hello Framework! A heuristic method for choosing front-end JavaScript frameworks," degree of Bachelor Independent thesis Basic level, Department of Information Technology, Uppsala University, Disciplinary Domain of Science and Technology, Mathematics and Computer Science, Suecia, 2021.

[28] R. Vyas, "Comparative Analysis on Front-End Frameworks for Web Applications," *International Journal for Research in Applied Science and Engineering Technology,* vol. 10, no. 7, pp. 298-307, 2022.

[29] K. Kambona, E. G. Boix, and W. De Meuter, "An evaluation of reactive programming and promises for structuring collaborative web applications," in *Proceedings of the 7th Workshop on Dynamic Languages and Applications*, 2013, pp. 1-9.

[30] S. Abrahamsson, "A model to evaluate front-end frameworks for single page applications written in JavaScript," degree of Master (Two Years) Independent thesis Advanced level, Department of Computer and Information Science, Linköping University, Suecia, 2023.

[31] H. M. Faruque, "To Build a Front-End Web Application Which JavaScript Framework is Trending Nowadays," Master Thesis for the Degree of Masters of Engineering, Department of Global Smart IT Convergence Engineering, University of Ulsan, Ulsan, Korea, 2022.

[32] J. Duvander and O. Romhagen, "What affects the choice of a JavaScript framework: Interviews with developers," Bachelor Independent thesis Basic level Jönköping University, School of Engineering, JTH, Computer Science and Informatics., Suecia, 2019.

[33] E. Saks, "JavaScript Frameworks: Angular vs React vs Vue," Bachelor Thesis, Business Information Technology, Haaga-Helia: University of Applied Sciences, Finlandia, 2019.

[34] P. D. Dutonde, S. S. Mamidwar, M. S. Korvate, S. Bafna, and P. Shirbhate, "Website Developmemt Technologies: A Review," *International Journal for Research in Applied Science and Engineering Technology,* vol. 10, no. 1, pp. 359-366, 2022.

[35] B. Quvvatov, "WEB FRONT-END AND BACK-END TECHNOLOGIES IN PROGRAMMING," *Theoretical aspects in the formation of pedagogical sciences,* vol. 3, no. 1, pp. 208-215, 2024.

[36] B. Wu, Y. Cao, and X. Liu, "Research and Analysis of Commonly Used Node. js Framework," in *2024 Prognostics and System Health Management Conference (PHM)*, 2024, pp. 104-107: IEEE.

[37] B. Zima and M. Barszcz, "Comparative analysis of Node. js frameworks," *Journal of Computer Sciences Institute,* vol. 30, pp. 26-30, 2024.

[38] E. Meijer and P. Drayton, "Static typing where possible, dynamic typing when needed: The end of the cold war between programming languages," 2004: Citeseer.

[39] M. Golec and M. Plechawska-Wójcik, "Comparative analysis of frameworks using TypeScript to build server applications," *Journal of Computer Sciences Institute,* vol. 23, pp. 128-134, 2022.

[40] stackshare. (2024, 1 de noviembre). *NestJS vs Yii | What are the differences?* Available: <https://stackshare.io/stackups/nestjs-vs-yii>

[41] K. Lei, Y. Ma, and Z. Tan, "Performance comparison and evaluation of web development technologies in php, python, and node. js," in *2014 IEEE 17th international conference on computational science and engineering*, 2014, pp. 661-668: IEEE.

[42] J. Thönes, "Microservices," *IEEE software,* vol. 32, no. 1, pp. 116-116, 2015.

[43] S. Kumar, "A Review on Client-Server based applications and research opportunity," *International Journal of Recent Scientific Research,* vol. 10, no. 7, pp. 33857-3386, 2019.

[44] L. B. Jiménez, "Propuesta de arquitectura de software para el producto “Mis Mejores Cuentos”," Trabajo de Diploma para optar por el título de Ingeniero en Ciencias

Informáticas, Universidad de las Ciencias

Informáticas, La Habana, Cuba, 2011.

[45] T. Alfers, G. Gittler, E. Ulitzsch, and S. Pohl, "Assessing the Speed–Accuracy Tradeoff in Psychological Testing Using Experimental Manipulations," *Educational and Psychological Measurement,* vol. 85, no. 2, pp. 357-383, 2025.