ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

SISTEMA DE PREPLANIFICACIÓN DE ASIGNATURAS PARA LA FACULTAD DE INGENIERÍA DE SISTEMAS

HERRAMIENTAS QUE SOPORTEN LA ENTREGA CONTINUA DE INCREMENTOS FUNCIONALES DEL SISTEMA DE PREPLANIFICACIÓN DE ASIGNATURAS PARA LA FIS

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SOFTWARE

CHRISTIAN ALEJANDRO MORÁN GALARZA

christian.moran@epn.edu.ec

DIRECTOR: JULIO CÉSAR SANDOBALÍN GUAMÁN

julio.sandobalin@epn.edu.ec

DMQ, febrero 2024

CERTIFICACIONES

Yo, CHRISTIAN ALEJANDRO MORÁN GALARZA declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

_____

CHRISTIAN ALEJANDRO MORÁN GALARZA

Certifico que el presente trabajo de integración curricular fue desarrollado por CHRISTIAN ALEJANDRO MORÁN GALARZA, bajo mi supervisión.

_____

JULIO CÉSAR SANDOBALÍN GUAMÁN

DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

CHRISTIAN ALEJANDRO MORÁN GALARZA

JULIO CÉSAR SANDOBALÍN GUAMÁN

DEDICATORIA

Deseo dedicar la culminación de esta etapa de mi vida a mi familia, quienes representan el corazón de mis acciones y decisiones. A mis compañeros de clase, quienes se transformaron en amigos invaluables en el transcurso de este camino. Mi agradecimiento se extiende a todas las personas que tuve el honor de conocer en este largo trayecto, y que contribuyeron a moldear la persona que soy hoy. Quiero reservar un lugar especial en esta dedicatoria para mi madre, cuyo cuidado siempre me ha rodeado, y a mi padre, quien, aunque físicamente ya no está, sigue siendo una parte fundamental en mi vida.

AGRADECIMIENTO

Deseo expresar mi profundo agradecimiento a un grupo invaluable de personas que han sido mi pilar durante esta travesía. En primer lugar, a mi madre, cuyos esfuerzos sobrehumanos han permitido que pueda dedicarme a mis estudios, espero algún día estar al nivel de tu sacrificio. A mis hermanos, quienes son mi constante motivación para crecer como persona y para sacrificar lo necesario por el bienestar familiar. Mi gratitud se extiende a aquellos individuos que han dejado una huella imborrable en mi corazón, brindándome su apoyo incluso en los momentos más difíciles. También deseo reconocer y agradecer a los excelentes docentes que tuve el privilegio de conocer en este camino, ya que me mostraron que es posible convertir una pasión en una profesión.

ÍNDICE DE CONTENIDO

RESUMEN

Este documento presenta un estudio de las prácticas de DevOps que soportan la entrega continua de incrementos funcionales del sistema de preplanificación de asignaturas del período académico ordinario para la Facultad de Ingeniería de Sistemas FIS. Se explora la implementación y adopción de la plataforma Azure DevOps que desempeñan un papel fundamental al facilitar la integración y despliegue continuos de los artefactos del producto software. En este contexto, se examinan todas las etapas necesarias para llevar a cabo una entrega continua de incrementos del producto software. Se puso énfasis en la inclusión de buenas prácticas establecidas por el equipo, asegurando que cada componente se integre fácilmente y contribuya de manera significativa al correcto funcionamiento del sistema. Además, se resalta la importancia de la automatización de pruebas en los pipelines de integración y despliegue continuos. Así, se garantiza la experiencia de usuario al obtener un producto software funcional y se minimizan los riesgos asociados a posibles fallos. El trabajo culmina con la exitosa liberación de una versión operativa del sistema de preplanificación de asignaturas para la FIS.

PALABRAS CLAVE: Integración Continua, Despliegue Continuo, Azure DevOps, Pipeline.

ABSTRACT

The document presents a study on DevOps practices supporting the continuous delivery of functional increments for the academic course planning system at the Facultad de Ingeniería de Sistemas FIS. The implementation and adoption of the Azure DevOps platform are explored, playing a pivotal role in enabling continuous integration and deployment of software artifacts. Within this framework, all necessary stages for achieving continuous delivery of software increments are examined, with a focus on incorporating established best practices by the team. Emphasis is placed on ensuring seamless integration of each component and its meaningful contribution to the system's proper functioning. Additionally, the significance of test automation in continuous integration and deployment pipelines is underscored, aiming to ensure a user-friendly software product while mitigating risks associated with potential failures. The study culminates in the successful release of an operational version of the course planning system for the FIS.

KEYWORDS: Continuous Integration, Continuous Deployment, Azure DevOps, Pipeline.

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

La Facultad de Ingeniería de Sistemas FIS realiza una encuesta de preplanificación para conocer el número de estudiantes que están interesados en matricularse en determinadas asignaturas el próximo Período Académico Ordinario PAO. Para realizar la encuesta se utiliza la herramienta Microsoft Forms para configurar las preguntas y recolectar las respuestas. Microsoft Forms es una herramienta que no tiene la capacidad de realizar validaciones en el cuestionario, ni en las preguntas. En consecuencia, la encuesta de preplanificación es propensa a errores y no brinda una información precisa.

La concepción de este componente surge de la necesidad de optimizar el tiempo de entrega del sistema de preplanificación de asignaturas de la Facultad de Ingeniería de Sistemas. El objetivo del sistema de preplanificación es ofrecer una solución eficiente y precisa, considerando prerrequisitos, correquisitos y pisos en las encuestas. Obteniendo una intención de matrícula de los estudiantes más realista.

La integración y despliegue continuos requieren una sinergia efectiva entre herramientas. Su objetivo es respaldar la entrega continua de incrementos funcionales de software. Este componente explora e implementa una serie de herramientas que respaldan la entrega continua de incrementos funcionales en el desarrollo ágil del producto software. El análisis comparativo de opciones en el mercado permitió seleccionar la herramienta más adecuada para el proyecto, basándose en criterios definidos por el equipo de trabajo para cumplir con las demandas de escalabilidad, automatización, integración y colaboración del sistema.

DevOps es un movimiento que promueve la continua colaboración entre los desarrolladores y el equipo de operaciones. En este contexto, DevOps promueve las prácticas de integración, despliegue y entrega continua, fundamentales en la administración eficiente de incrementos funcionales de software. La integración continua fusiona código regularmente para detectar problemas de forma temprana y asegurar coherencia. El despliegue continuo automatiza la implementación de nuevas funcionalidades, agilizando la disponibilidad del producto para los usuarios finales. Por otra parte, la entrega continua garantiza la preparación constante de incrementos funcionales, proporcionando flexibilidad en un entorno ágil. Estas prácticas son pilares para una entrega fluida, eficiente y confiable en el sistema de preplanificación.

Azure DevOps es una plataforma tecnológica que permite implementar las prácticas de DevOps y, por lo tanto, desempeña un papel esencial en este componente. Sus herramientas como Wikis, Boards, Repos y Pipelines, facilita la planificación colaborativa, el seguimiento de tareas, el control de versiones y la automatización del ciclo de vida del

software. Las Wikis permitieron una documentación compartida, mientras que Boards agiliza la gestión de tareas y seguimiento del progreso del desarrollo del producto software. Repos proporciona control de versiones y colaboración en el código fuente, y Pipelines automatiza etapas como la integración, pruebas y despliegue, optimizado la eficiencia del proceso del desarrollo de la aplicación web.

Mediante Azure DevOps, se logró implementar de manera efectiva las prácticas de integración y despliegue continuos mediante pipelines. La automatización permitió la orquestación de las etapas cruciales del proceso de desarrollo, desde la integración del código hasta su despliegue final. La configuración intuitiva de los pipelines facilitó la creación de flujos de trabajo personalizados que aseguraron la coherencia, eficiencia y la entrega continua. Cabe destacar que, aunque no se incluyeron Test Plans de Azure DevOps en el alcance del componente, se realizaron pruebas integrales como parte del ciclo de vida de CI/CD, contribuyendo a un sistema con comprobada confiabilidad y alta calidad.

1.1 Objetivo general

Implementar un encadenamiento de herramientas que soporten la entrega continua de incrementos funcionales de software en el desarrollo del sistema de preplanificación de asignaturas del período académico ordinario para la FIS.

1.2 Objetivos específicos

1. Analizar herramientas que apoyen las prácticas de integración y despliegue continuos de incrementos funcionales de software.
2. Implementar pipelines con herramientas que soporten las prácticas de integración y despliegue continuos de incrementos funcionales de software.
3. Verificar la implementación de los pipelines de integración y despliegue continuos durante el desarrollo del sistema de preplanificación de asignaturas del período académico ordinario para la FIS.

1.3 Alcance

La importancia de este componente es desarrollar el análisis y ejecutar la implementación de las prácticas de integración, entrega y despliegue continuos de incrementos funcionales de software. Para lograrlo, se eligió la plataforma Azure DevOps, donde se utilizó herramientas tales como Repos y Pipelines. Estas herramientas forman el núcleo de las prácticas de integración, entrega y despliegue continuos. En el desarrollo ágil de las asignaturas del período académico ordinario para la FIS se verificó el correcto funcionamiento de las prácticas de DevOps. Por lo tanto, cuando se realizó un cambio en el código fuente del proyecto, Azure DevOps automatiza la ejecución de los pipelines de integración y despliegue, permitiendo así una gestión fluida y eficiente de los procesos en el desarrollo de software.

El resultado más notable del componente reside en la automatización de tareas rutinarias, liberando tiempo valioso y asegurando una mayor eficiencia en el proceso de desarrollo de software al realizar un cambio y publicarlo. La implementación exitosa de todas las etapas del ciclo de vida del CI/CD garantiza una gestión eficiente y fluida de los procesos clave en el desarrollo de software. La implementación de Azure DevOps y de las herramientas que lo componen proporcionó soporte al desarrollo al sistema de preplanificación de asignaturas para la FIS.

1.4 Marco teórico

DevOps

DevOps es un enfoque integral que prioriza la colaboración y automatización para lograr un ciclo de desarrollo más efectivo y fiable (Jabbari et al., 2016).DevOps se caracteriza por su "Cultura de Colaboración", que fomenta la comunicación fluida entre los equipos de desarrollo y operaciones, permitiendo una entrega de software más ágil y confiable. Los equipos trabajan juntos desde el diseño hasta la implementación y el mantenimiento del software.

El paradigma DevOps se basa en la automatización, impulsando flujos de trabajo automáticos para tareas repetitivas y propensas a errores, como la compilación, pruebas y despliegue. Esta automatización agiliza procesos, reduce fallos humanos, mejora la calidad del software y eficientiza el ciclo de desarrollo. DevOps también se sustenta en la monitorización constante, retroalimentación y mejora continua. Equipos identifican obstáculos y deficiencias en el proceso de desarrollo y entrega, para mejorar la calidad del software y colaboración interdisciplinaria.

Principios, Prácticas y Herramientas

Los **principios** fundamentales de DevOps son la colaboración para automatizar tareas repetitivas, monitorización constante para retroalimentación y mejora continua, en la búsqueda de eficiencia en el ciclo de desarrollo y entrega de software. Estos principios impulsan la cultura de trabajo conjunto, agilidad y calidad en la industria del desarrollo de software.

Las **prácticas** de DevOps abarcan la Integración, Entrega y Despliegue Continuo que garantizan calidad y eficiencia en el desarrollo ágil de un producto software. La automatización de tareas como compilación, pruebas y despliegue agiliza procesos, minimizando errores humanos y asegurando software confiable. Además, la retroalimentación constante permite identificar y solventar obstáculos asegurando la satisfacción del cliente (Zhu et al., 2016).

Las **herramientas** que dan soporte a las prácticas de DevOps son cruciales para las prácticas ágiles. Estas herramientas abarcan desde la automatización de flujos de trabajo hasta la gestión de la infraestructura como código. Entre las herramientas más conocidas están Azure DevOps, Jenkins, Docker, y Kubernetes que facilitan colaboración, automatización y entrega eficiente de software al tiempo que mejoran la calidad y confiabilidad del proceso.

En cuanto a las pruebas, herramientas como Selenium, JUnit y TestNG se destacan por su capacidad para realizar pruebas automatizadas, garantizando una validación exhaustiva de la funcionalidad del software. Estas herramientas permiten la creación de casos de prueba y la ejecución automatizada, contribuyendo así a la detección temprana de posibles problemas.

Para la gestión de la calidad del código, herramientas como SonarQube y ESLint ofrecen análisis estático del código fuente, identificando posibles problemas de calidad, vulnerabilidades y asegurando la coherencia con los estándares definidos.

En el ámbito de la monitorización y registro de aplicaciones, herramientas como Prometheus y ELK Stack (Elasticsearch, Logstash y Kibana) son esenciales. Estas permiten la recopilación, visualización y análisis de registros y métricas, facilitando la identificación y resolución proactiva de problemas en la aplicación en producción.

Cabe destacar que la elección de herramientas específicas puede variar según los requisitos y tecnologías utilizadas en cada proyecto, pero la combinación adecuada de estas herramientas garantiza un proceso robusto y eficiente.

Integración Continua (Continuous Integration)

La práctica de desarrollo de software denominada **integración continua** involucra a los desarrolladores en la fusión periódica de los ajustes realizados en el código en un repositorio central. Posteriormente, se ejecutan pruebas automáticas y se generan versiones.

Este concepto se centra mayormente en la fase de integración durante el proceso de lanzamiento del software, donde la automatización es un elemento clave. Los objetivos primordiales abarcan la detección y corrección acelerada de errores, la mejora en la calidad del software y la disminución del tiempo requerido para validar y liberar nuevas actualizaciones del sistema.

El proceso de "Build and Deployment" también conocido como **construcción y despliegue** es uno de los procesos más importantes. Este proceso asegura que las novedades y modificaciones implementadas por el equipo de desarrollo se integren de manera constante y confiable en el producto principal (Zúñiga-Prieto et al., 2017).

Es crucial seguir buenas prácticas como la automatización de pruebas, el establecimiento de entornos de integración y el monitoreo constante de los procesos. Los beneficios incluyen mayor calidad del software, ciclos de desarrollo más rápidos y colaboración mejorada entre equipos.

El **ciclo de integración continua** involucra un pipeline para completar su objetivo principal, conformada por una serie de etapas, estas se describen a continuación:

- **Análisis de código estático:** Detecta vulnerabilidades y asegura cumplimiento de estándares.
- **Compilación:** Transforma el código fuente en un artefacto ejecutable.
- **Pruebas unitarias:** Verifica unidades de código aisladas para identificar errores.
- **Implementación:** Genera código listo para desplegar en pruebas o producción.
- **Pruebas de integración:** Evalúa el artefacto ejecutable en entornos de prueba con conexiones externas.

Adicionalmente, la integración continua involucra la automatización del proceso de creación y actualización del código. En consecuencia, cada vez que se realicen cambios en el repositorio de código, se lleva a cabo automáticamente la construcción y se despliega una versión actualizada del producto.

Despliegue Continuo (Continuous Deployment)

El **despliegue continuo** se fundamente en la automatización para facilitar la implementación de cambios en el entorno de producción. Esta práctica impulsa la agilidad, la eficacia y la excelencia en la distribución de software, otorgando a los equipos la capacidad de mantenerse a la vanguardia en un mercado en constante evolución (Humble & Farley, 2010).

En esta perspectiva, siempre que se realice una modificación en el código y este pase las pruebas automatizadas, se efectúa su despliegue automático en el entorno de producción sin requerir intervención manual.

Las etapas de construcción, pruebas y despliegue son coordinadas a través de la automatización, lo que resulta en una serie de beneficios esenciales, tales como la distribución frecuente, la retroalimentación inmediata, una mayor visibilidad y la capacidad de adaptación a las demandas cambiantes, los cuales son especialmente vitales en entornos ágiles y competitivos.

Entrega Continua (Continuous Delivery)

La **entrega continua** es una práctica fundamental en DevOps que busca la eficiencia y mejora del proceso de liberación de software. Esta práctica tiene como principal propósito la entrega constante y confiable de incrementos funcionales al sistema. La liberación regular de pequeñas actualizaciones de software, mitigando el riesgo de errores y facilitando a una respuesta ágil a los cambios en requisitos o condiciones del mercado.

Se nutre de la integración continua, práctica donde el código se fusiona y verifica automáticamente en busca de inconvenientes. Posteriormente, los cambios son sometidos a pruebas y desplegados en un entorno de pruebas que replica el entorno de producción. Esta secuencia garantiza un funcionamiento coherente y libre de problemas del software. Adoptar la práctica de entrega continua permite al equipo de desarrollo proporcionar valor de manera rápida y continua, acortando los ciclos de desarrollo y permitiendo una adaptación ágil a las demandas de usuarios y mercados (Chen, 2015).

La principal diferencia entre la entrega y despliegue continuos radica en el momento en el cual se toma la decisión de implementar los cambios en el ambiente de producción. En la entrega continua, esta decisión es manual y en el despliegue continuo es automatizado. Ambas prácticas agilizan el proceso de desarrollo y liberación de software, reduciendo errores y acelerando la entrega de valor a los usuarios.

Infraestructura como Código

La práctica de infraestructura como código (IaC) es la piedra angular de DevOps para lograr una gestión eficiente y ágil de los entornos de desarrollo, prueba y producción. En lugar de configurar manualmente servidores y recursos de infraestructura, la IaC permite describir la infraestructura utilizando código, lo que brinda numerosos beneficios. Los equipos pueden definir y aprovisionar recursos de manera consistente y reproducible, reduciendo así errores humanos y desviaciones en la configuración.

A medida que los sistemas crecen, la gestión manual se vuelve inviable, pero la IaC facilita la escalabilidad al permitir la creación de plantillas y módulos reutilizables. Además, proporciona una forma documentada y auditada de administrar la infraestructura. El código describe claramente cómo se configuran y relacionan los componentes, lo que facilita el seguimiento de cambios, la revisión de la configuración y la identificación de posibles problemas.

En este contexto, la presencia del lenguaje YAML se vuelve crucial. Este lenguaje sigue una sintaxis sencilla y legible, especifica cómo se debe construir y configurar la infraestructura. Al emplear un archivo de configuración con YAML, se facilita la colaboración entre equipos, se documentan las configuraciones de manera clara y se promueve la consistencia en la gestión de la infraestructura, contribuyendo así a un flujo de trabajo fluido y confiable en el marco del desarrollo y despliegue de software.

Pruebas de Software

Las pruebas de software tienen dos objetivos claves. Primero, demuestran que el software cumple con los requisitos del cliente. Segundo, buscan identificar situaciones donde el comportamiento del software es incorrecto o no se ajusta a las especificaciones, revelando posibles defectos.

Las pruebas no garantizan la ausencia total de defectos ni un comportamiento perfecto del software. Forman parte del proceso más amplio de verificación y validación, que se centra en construir el producto correcto (validación) y construirlo correctamente (verificación), asegurando que los requerimientos se alineen con las expectativas del cliente.

Además de las pruebas, la verificación y validación involucran inspecciones y revisiones del software, que abarcan requerimientos, diseño, código y pruebas propuestas. Aunque útiles para descubrir errores, las inspecciones no reemplazan las pruebas, ya que no son adecuadas para errores derivados de interacciones inesperadas, problemas temporales o de rendimiento. Estos procesos conjuntos aseguran un producto que cumple con las expectativas y está libre de defectos reconocibles (Sánchez Peño, 2015).

Pruebas Unitarias

Las pruebas unitarias constituyen un tipo específico de evaluación en el desarrollo de software, focalizado en analizar la funcionalidad de una aplicación, que asiste a los desarrolladores en la detección temprana de errores en el código. Esto permite ofrecer un producto de óptima calidad a los usuarios finales. Esta práctica forma parte esencial del proceso de desarrollo de software y ejerce un impacto significativo en la calidad del código generado.

Este enfoque de pruebas se concentra en examinar fragmentos de código para garantizar que los resultados actúen de acuerdo con el diseño y los requerimientos establecidos. Cada prueba unitaria analiza una única funcionalidad dentro de una unidad, y está diseñada para ser independiente de otras pruebas y del entorno en el que se ejecutan.

Pruebas End-to-End

Las pruebas end-to-end surgen para solucionar problemas de corrección y rendimiento en sistemas distribuidos. Estas pruebas, consideradas funcionales de caja negra, emulan el comportamiento real del sistema para verificar su funcionamiento.

Un concepto clave en estas pruebas es el "trazado", que visualiza eventos y relaciones entre componentes, proporcionando una representación visual. Además, generan métricas como carga de trabajo, uso de recursos y tiempo, permitiendo comprender la operación del sistema, detectar anomalías en tiempo real y analizar fallos potenciales para mejorar la calidad y el rendimiento de sistemas distribuidos.

Pruebas de Carga y Estrés

Las pruebas de carga y estrés tienen como objetivo principal analizar el rendimiento y la capacidad de una aplicación en diversos niveles de demanda y estrés (Serna M. et al., 2019).

Las **pruebas de carga** se implementan con el fin de examinar el comportamiento de una aplicación sometida a una carga de trabajo habitual o incluso superior a la esperada bajo condiciones usuales. El propósito primordial es determinar si la aplicación puede manejar sin declives notables en su rendimiento la cantidad estimada de usuarios y transacciones. En este proceso, se monitorizan indicadores como el tiempo de respuesta y la capacidad de procesamiento para detectar posibles cuellos de botella y áreas vulnerables en la aplicación.

En contraste, las **pruebas de estrés** llevan a la aplicación a situaciones límite o extremas para evaluar su comportamiento en contextos de alta presión. Esto implica incrementar la carga más allá de lo comúnmente esperado en el uso cotidiano.

El propósito consiste en observar cómo la aplicación reacciona en situaciones atípicas, como picos de tráfico o demandas intensas, y si es capaz de recuperarse apropiadamente tras episodios de fallos.

Azure DevOps

Azure DevOps es una plataforma tecnológica que facilita la implementación de procesos claves como la integración, despliegue y entrega continuos de incrementos funcionales de software. Al mismo tiempo que el monitoreo y retroalimentación para realizar una mejora continua. Esta herramienta ofrece integración con herramientas que analizan el código, verifican calidad, seguridad y aprovisionamiento de infraestructura.

Wikis

Las **Wikis** de Azure ofrecen un espacio colaborativo para la documentación y el intercambio de conocimientos entre los miembros de un equipo. Facilitando la creación de documentación actualizada, tutoriales, guías y cualquier otro recurso relevante que contribuya a una comunicación más efectiva (Rossberg, 2019).

Boards

Ofrece tableros visuales que permiten una organización efectiva del trabajo en el desarrollo ágil de software, destacando especialmente en la aplicación del método Kanban. Al proporcionar tableros personalizables, asignación dinámica de tareas y notificaciones en tiempo real, se potencia la toma de decisiones ágil y la colaboración eficiente entre los miembros del equipo. Los informes generados ofrecen una visión detallada del estado del proyecto, brindando valiosa información para optimizar la planificación y aumentar la productividad en el proceso de desarrollo.

Backlogs

Los Backlogs en Azure DevOps es donde se crean y gestionan los elementos de trabajo que forman parte del proyecto. Aquí, los equipos pueden definir historias de usuario con su respectivo tamaño, tareas, y otros requisitos. Los elementos de trabajo, conocidos dentro de la herramienta como Work Items, se pueden priorizar y asignar a Sprints específicos para su implementación.

Sprints

Representan los períodos de tiempo definidos en los que se planifica y ejecuta el trabajo. Los elementos de trabajo del Backlog se seleccionan para su inclusión en un Sprint y se mueven a la vista del tablero de trabajo por sprint. Durante un Sprint, el equipo trabaja en la implementación de las tareas, con el objetivo de completarlas antes de que termine el Sprint.

Repos

Los repositorios de Azure DevOps, conocidos como **Repos**, gestionan el código fuente y permiten múltiples configuraciones por proyecto. Además de alojar el código de manera segura, ofrecen herramientas de colaboración, seguimiento de cambios y revisión de código. Esto fomenta la contribución conjunta y la garantía de calidad en el software.

Pipelines

Los pipelines automatizan la construcción, pruebas y despliegue de software. A través de, una configuración relativamente sencilla, permiten adaptarse a diferentes necesidades de desarrollo y proporciona una plataforma integral para la orquestación y ejecución de tareas cruciales en el ciclo de vida del software (Soh et al., 2020).

Es importante mencionar el concepto de un **pipeline**, conocido como la interconexión de múltiples etapas, cada una con un propósito específico que contribuye al enriquecimiento de las funcionalidades del producto. Es importante señalar que no hay un estándar establecido para la creación de un pipeline, lo que brinda al encargado de su desarrollo la flexibilidad de simplificar la adopción e implementación de prácticas de integración, entrega y despliegue continuo de acuerdo con su enfoque (Soh et al., 2020).

Por otro lado, las **Releases** se utilizan para coordinar y gestionar la implementación de software en diferentes entornos, como desarrollo, pruebas y producción. Las Releases permiten un mayor control y visibilidad a través de estos entornos, lo que garantiza una entrega confiable y coherente.

Los Pipelines se centran en la automatización del proceso de construcción y despliegue, mientras que las Releases se utilizan para coordinar y administrar la implementación de software en diversos entornos como desarrollo, producción y pruebas.

Test Plans

Los **Test Plans** permiten definir pruebas y agrupar casos según características específicas, brindando información detallada sobre el progreso.

Artifacts

Finalmente, Azure DevOps Artifacts gestiona paquetes esenciales para la implementación de software, administrando artefactos y tipos de paquetes.

Agilismo

El agilismo es un conjunto de marcos, metodologías y prácticas que se centran en la adaptabilidad, colaboración y entrega constante de valor. Se basa en el manifiesto ágil, un conjunto de principios que enfatizan la satisfacción del cliente, la respuesta al cambio, la comunicación efectiva y la entrega incremental. El manifiesto ágil se originó como respuesta a los enfoques tradicionales de gestión de proyectos, buscando promover un enfoque más flexible y orientado al cliente.

El **manifiesto ágil** se compone de cuatro valores fundamentales: valorar a los individuos y las interacciones por encima de los procesos y las herramientas; priorizar el software funcional sobre la documentación exhaustiva; colaborar con los clientes más que negociar contratos y responder al cambio en lugar de seguir un plan rígido. Estos valores se reflejan en los doce principios ágiles, que enfatizan la satisfacción del cliente, la entrega frecuente de software, la colaboración cercana entre equipos y clientes, la atención continua a la calidad técnica y la adaptación ágil a los cambios en los requisitos (Beck et al., 2001).

Los **principios del agilismo** guían el enfoque hacia una gestión de proyectos y desarrollo de software más efectiva. El primero destaca la satisfacción del cliente a través de entregas tempranas y continuas de software funcional, estableciendo una comunicación constante con el cliente para adaptar los requisitos. El segundo principio resalta la importancia de la entrega constante de software, priorizando incrementos pequeños y frecuentes que generen valor.

El tercer principio promueve la colaboración activa entre los equipos de desarrollo y los usuarios, garantizando que las soluciones aborden las necesidades reales del cliente. El cuarto principio insta a los equipos a valorar los cambios en los requisitos incluso en etapas avanzadas del desarrollo, abrazando la flexibilidad ante las demandas cambiantes. El quinto principio se centra en mantener una atención constante a la calidad técnica y el diseño, evitando la acumulación de deuda técnica.

El sexto principio aboga por la simplicidad en el diseño y la implementación, evitando la creación de soluciones excesivamente complejas. El séptimo principio destaca la importancia de empoderar a los equipos para la toma de decisiones, permitiendo una mayor autonomía y responsabilidad. El octavo principio enfatiza la colaboración entre equipos y personas, priorizando la comunicación directa para resolver problemas y compartir conocimientos (Letelier & Penadés, 2017).

En resumen, los principios del agilismo se centran en la satisfacción del cliente, la entrega continua de valor, la colaboración efectiva y la adaptación ágil a los cambios. Estos principios guían a los equipos y organizaciones en la búsqueda de una gestión de proyectos más flexible y orientada a resultados, promoviendo una cultura de mejora continua y adaptación en un entorno dinámico.

Scrum

“Su nombre no es una sigla, sino un término aplicable al rugby” (Zumba,2018, p12). Scrum se presenta como un enfoque ágil que ha demostrado su eficacia en la entrega de valor a través de un proceso iterativo e incremental. Este marco de trabajo, que tuvo su origen en la década de 1990, ha ganado notoriedad en diversos sectores debido a su capacidad para enfrentar de manera efectiva desafíos complejos y cambiantes. Scrum se fundamenta en pilares de transparencia, inspección y adaptación constante, y se caracteriza por sus roles definidos, eventos estructurados y artefactos esenciales.

Scrum se deriva de los principios ágiles, una filosofía que busca agilidad, flexibilidad y un enfoque centrado en el cliente. Estos principios se incorporan en Scrum a través de enfoques conceptuales, tales como:

Las **Entregas Continuas** en Scrum promueven la entrega de incrementos funcionales del producto en intervalos regulares, permitiendo obtener retroalimentación temprana y ajustar enfoques según sea necesario.

La **Colaboración Activa** permite que los equipos Scrum trabajan en estrecha colaboración con los stakeholders aseguren una comprensión completa de los requisitos y una alineación con las expectativas del cliente.

La **Adaptabilidad** en Scrum permite adaptarse a los cambios en los requisitos y condiciones, posibilitando que los equipos respondan rápidamente a las cambiantes demandas del mercado y los stakeholders del proyecto.

Scrum es un marco de trabajo ágil ampliamente utilizado en la gestión de software y otras áreas. En Scrum, el trabajo se organiza en iteraciones llamadas "sprints", que suelen tener una duración de 2 a 4 semanas. Cada sprint tiene un objetivo claro y produce un incremento del producto potencialmente entregable.

Las reuniones diarias, llamadas Daily Scrum, permiten al equipo compartir el progreso y corregir los problemas. Las reuniones de Planificación de Sprint y Revisión de Sprint ayudan a definir el objetivo del sprint y evaluar el incremento del producto software, respectivamente.

Scrum enfatiza la retroalimentación constante y la mejora continua. Después de cada sprint, se lleva a cabo una Retrospectiva para identificar oportunidades de mejora en el proceso. En resumen, Scrum ofrece un marco de trabajo ágil para equipos que buscan flexibilidad, colaboración y adaptabilidad en sus proyectos.

Scrum maneja 3 **roles** claves: el Product Owner representa a los stakeholders y es responsable de definir el backlog del producto; el Scrum Master quien es facilitador y responsable del proceso Scrum; y finalmente el equipo de desarrollo que son los responsables de construir el incremento funcional del producto software durante cada sprint (Balarezo & Torres, 2022).

Los artefactos en Scrum son elementos esenciales que ayudan a facilitar la planificación, seguimiento y entrega de valor en los proyectos ágiles. Por ejemplo, el Product Backlog es una lista priorizada de elementos de trabajo que representan las funcionalidades, mejoras y correcciones que se desean en el producto. Este artefacto es responsabilidad del Product Owner, quien lo mantiene actualizado y lo utiliza como base para la planificación.

El Sprint Backlog es una selección de elementos del Product Backlog que el equipo de desarrollo se compromete a completar durante un sprint. Contiene tareas más detalladas y es gestionado por el propio equipo para asegurar que el objetivo del sprint se alcance.

El Incremento es el resultado del trabajo del Equipo de Desarrollo al finalizar un sprint. Es un producto potencialmente entregable y puede incluir nuevas funcionalidades, mejoras y correcciones. Cada incremento es un paso hacia la realización del objetivo general del producto.

En síntesis, los artefactos, prácticas y eventos de Scrum son herramientas clave que permiten la organización y la transparencia en la planificación y ejecución de proyectos ágiles, asegurando que el valor se entregue de manera efectiva y constante.

2 METODOLOGÍA

Comparación de Herramientas

Con el propósito de lograr una entrega continua de incrementos funcionales y una gestión eficiente del sistema de preplanificación para la FIS, se llevó a cabo una comparación entre diversas herramientas disponibles en el mercado, que respalden la integración y el despliegue continuo.

Este proceso de análisis permitió identificar y seleccionar las herramientas más adecuadas para cumplir con los objetivos y requerimientos de este componente. El presente apartado detalla la metodología empleada para evaluar y elegir la herramienta que formará la base de nuestro enfoque de automatización.

Inicialmente, se abordó la tarea de definir las necesidades específicas del proyecto. Se consideraron aspectos tales como las tecnologías que se emplearían para la gestión del marco Scrum y la implementación tanto del Back End como del Front End con sus respectivas tecnologías. Se otorgó una importancia primordial a la curva de aprendizaje necesaria para la implementación de los pipelines adecuados para nuestro proyecto, tomando en cuenta el nivel de pericia que tenían los miembros del equipo de desarrollo.

Por lo tanto, se procedió a buscar las herramientas más populares en el mercado. Luego de una extensa investigación, se elaboró un cuadro comparativo con aquellas herramientas que presentaban las características más favorables. La **Tabla 1.** Comparativa de las Curvas de Aprendizaje de Herramientas para Creación de Pipelines CI/CD detalla la comparación de las curvas de aprendizaje de cada herramienta, ya que esto influiría en la rapidez con la que el equipo de desarrollo podría adaptarse a la tecnología y aprovecharla al máximo.

Tabla 1. Comparativa de las Curvas de Aprendizaje de Herramientas para Creación de Pipelines CI/CD

Criterios / Herramientas	Azure DevOps	Jenkins	GitLab CI/CD	CircleCI
Curva de aprendizaje	Moderada, especialmente para usuarios nuevos en Azure y sus servicios.	Variable, puede ser pronunciada debido a la configuración y scripting requeridos.	Moderada, la interfaz es amigable, pero la configuración avanzada puede requerir tiempo.	Moderada, la configuración inicial puede ser sencilla, pero características avanzadas pueden ser complejas.

Mientras tanto, en la Tabla 2. Comparativa de Herramientas para Creación de Pipelines CI/CD se compararon criterios como el tipo, hosting, licencia, soporte del sistema de control de versiones (VCS), lenguaje de definición del pipeline, orquestación de contenedores y soporte de diferentes plataformas de desarrollo.

Tabla 2. Comparativa de Herramientas para Creación de Pipelines CI/CD

Criterios / Herramientas	**Azure DevOps**	**Jenkins**	**GitLab CI/CD**	**CircleCI**
Tipo	Plataforma completa de ALM y CI/CD	Automatización de CI/CD	Integrado con plataforma de desarrollo	Automatización de CI/CD
Hosting	En la nube (Azure) o local	En la nube o local	En la nube o local	En la nube
Tipo de licencia	Freemium (con limitaciones)	Open source	Freemium (con limitaciones)	Freemium (con limitaciones)
Soporte de VCS	Sí	Sí	Sí	No
Lenguaje del pipeline	Archivos YAML o Diseñador visual	Archivos Groovy	Archivos YAML	Archivos YAML
Orquestación de contenedores	Integración con Docker y Kubernetes	Integración con Docker y Kubernetes	Integración con Docker y Kubernetes	Integración con Docker y Kubernetes
Soporte de plataformas	Amplio soporte multiplataforma	Amplio soporte multiplataforma	Amplio soporte multiplataforma	Amplio soporte multiplataforma

Como resultado de esta comparativa, inicialmente se optó por seleccionar Jenkins, ya que cumplía con los criterios del equipo de desarrollo y además era una herramienta totalmente gratuita. Sin embargo, tras varios intentos insatisfactorios en la configuración de los pipelines, tanto para el Back End como para el Front End, se convocó una reunión para abordar los problemas experimentados con la herramienta y tomar una decisión al respecto.

La evaluación de los inconvenientes presentados con Jenkins reveló que la curva de aprendizaje para las tecnologías de Back End y Front End resultó ser más desafiante de lo esperado. A pesar de los intentos, la configuración de los pipelines no fue completada. Como resultado, se tomó la decisión de migrar hacia Azure DevOps.

Azure DevOps fue elegido debido a su integración nativa en la creación de pipelines y su sólido soporte para el marco Scrum, abordando dos aristas importantes del sistema. La plataforma facilita la automatización y gestión de procesos, reduciendo la complejidad que encontramos con Jenkins. Esta transición nos permitió alcanzar una entrega y despliegue

continuos más eficientes y fluidos, con una curva de aprendizaje más adecuada para el tiempo en el que el sistema debía ser implementado.

Para concluir, la complejidad de la curva de aprendizaje de Jenkins para nuestras tecnologías de Back End y Front End fue un factor determinante en esta elección. La plataforma Azure DevOps fue preferida por su integración fluida en la creación de pipelines y respaldo al marco Scrum. Este cambio nos brinda la oportunidad de automatizar y gestionar procesos de manera más efectiva, superando las dificultades encontradas con Jenkins y mejorando la capacidad del equipo para lograr una entrega y despliegue continuos de manera más ágil y eficiente.

Configuración Inicial de Azure DevOps

La configuración de Azure DevOps involucró diversos pasos, desde la creación de la organización hasta la gestión de usuarios. Se comenzó creando la organización con el nombre único "TitulacionFisEPN" y seleccionando la ubicación, en este caso, "Brazil South". En la **Figura 1** se aprecia la organización configurada.

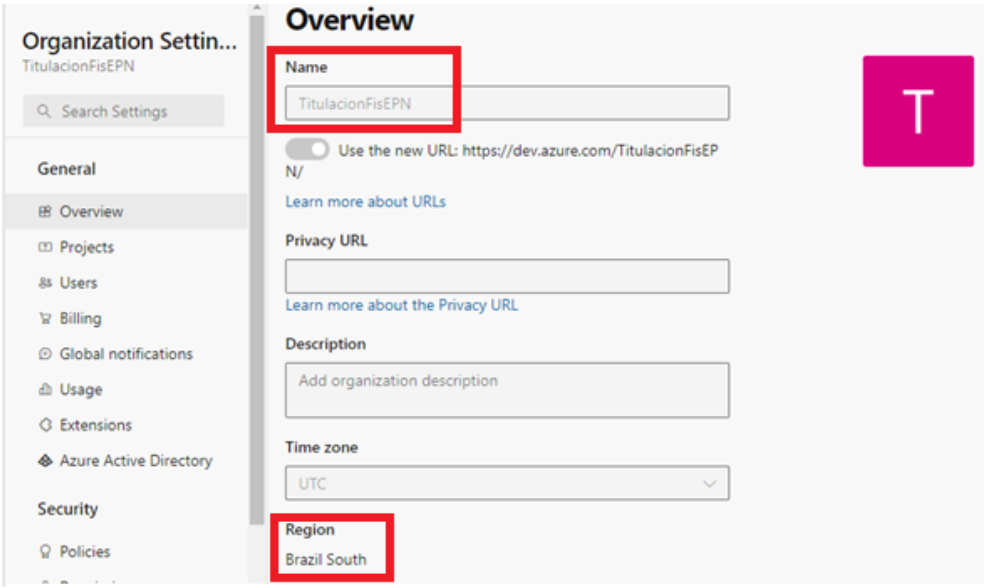


Figura 1. Organización configurada en Azure DevOps

Luego, se procedió a configurar el proyecto, asignando un nombre exclusivo, descripción, visibilidad y sistema de control de versiones. La **Figura 2** muestra el proyecto configurado dentro de la organización.

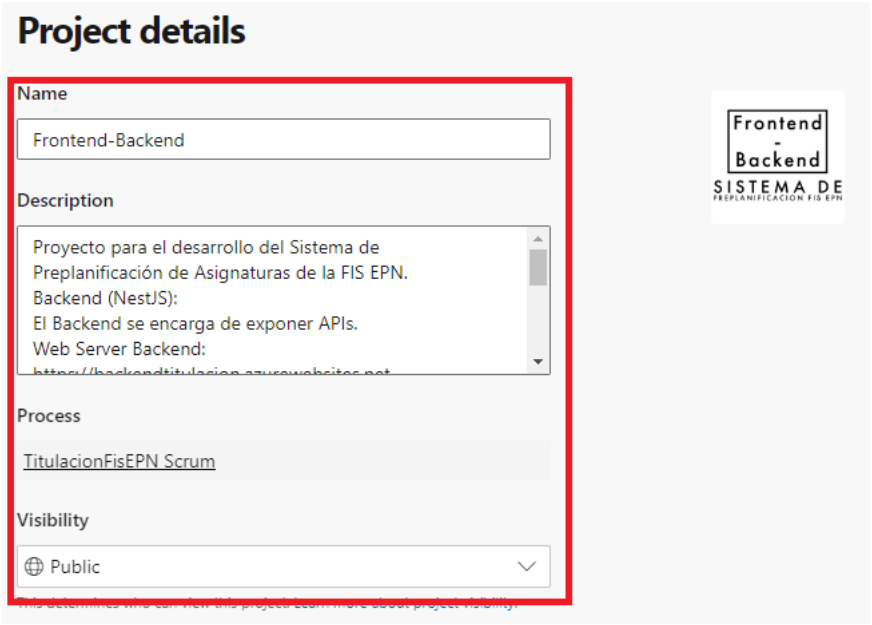


Figura 2. Proyecto configurado en la organización

Finalmente, se gestionaron los usuarios, incluyendo los correos de los miembros del equipo de desarrollo: César Santacruz, Christian Morán, Daniel Velastegui y David Aguirre. También se añadió al Scrum Master, PhD. Carlos Iñiguez, y al Product Owner, PhD. Julio Sandobalin. La **Figura 3** detalla los usuarios configurados en el proyecto con sus niveles de acceso respectivos, niveles asignados para evitar cambios.

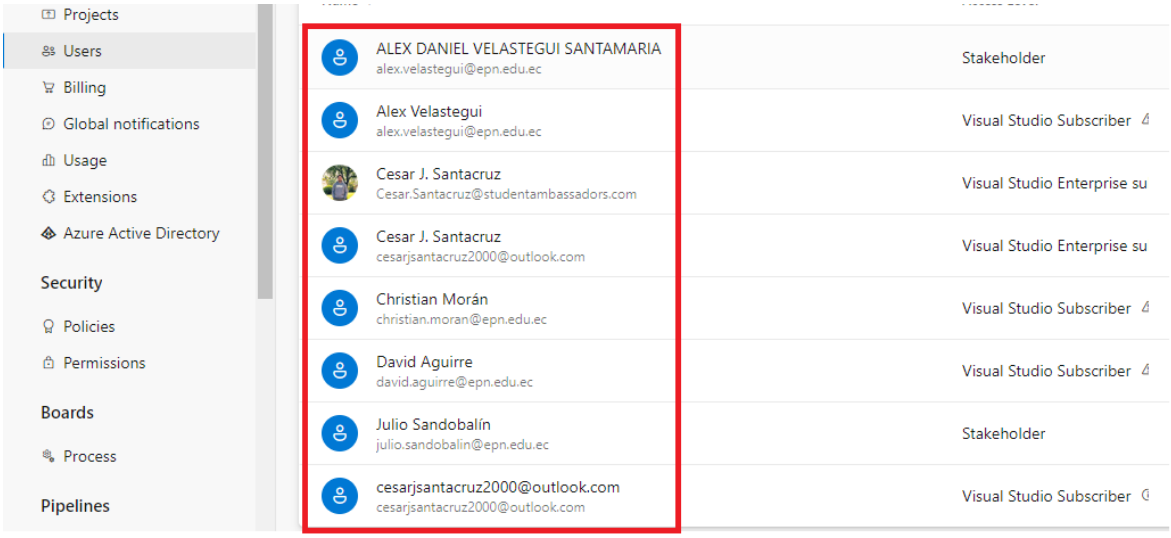


Figura 3. Usuarios configurados en la organización

Configuración del Sistema de Control de Versiones

Una de las ventajas al optar por Azure DevOps es su capacidad para establecer un espacio seguro y organizado destinado al código fuente. Junto con herramientas de colaboración y seguimiento de cambios, el equipo pudo crear ramas para cada versión, facilitando la colaboración en la codificación y asegurando una administración de las contribuciones. La integración con sistemas de control de versiones, como Git en este caso, simplificó tanto la gestión del código como la revisión de modificaciones.

En síntesis, Azure DevOps posibilitó la configuración de repositorios de manera eficiente tanto para el Back End como para el Front End. Esta configuración brindó las herramientas necesarias para manejar el código fuente, la implementación de procesos de control de versiones y la promoción de una colaboración en el desarrollo de software.

El equipo de desarrollo mantuvo un enfoque ágil en la construcción de los componentes claves del sistema. Los repositorios configurados por los equipos de Front End y Back End se ilustran en las **Figura 4** y **Figura 5**, donde se destacan los archivos. yml esenciales para una ejecución exitosa de CI/CD.

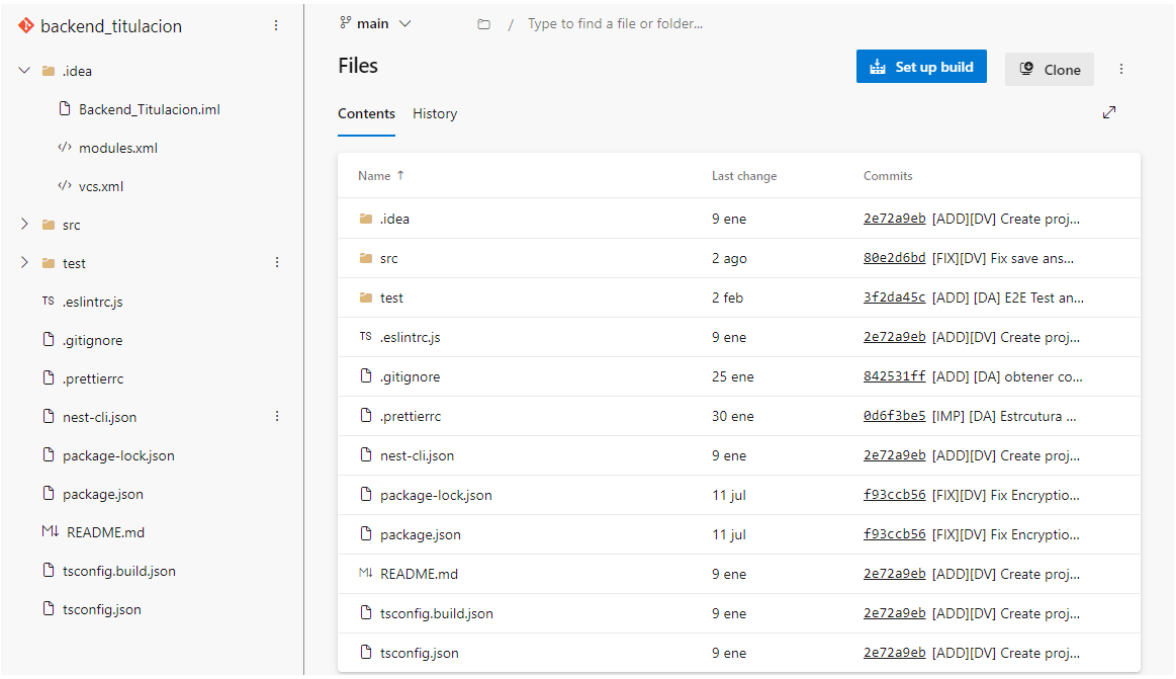


Figura 4.Repositorio para el Back End configurado en Azure DevOps

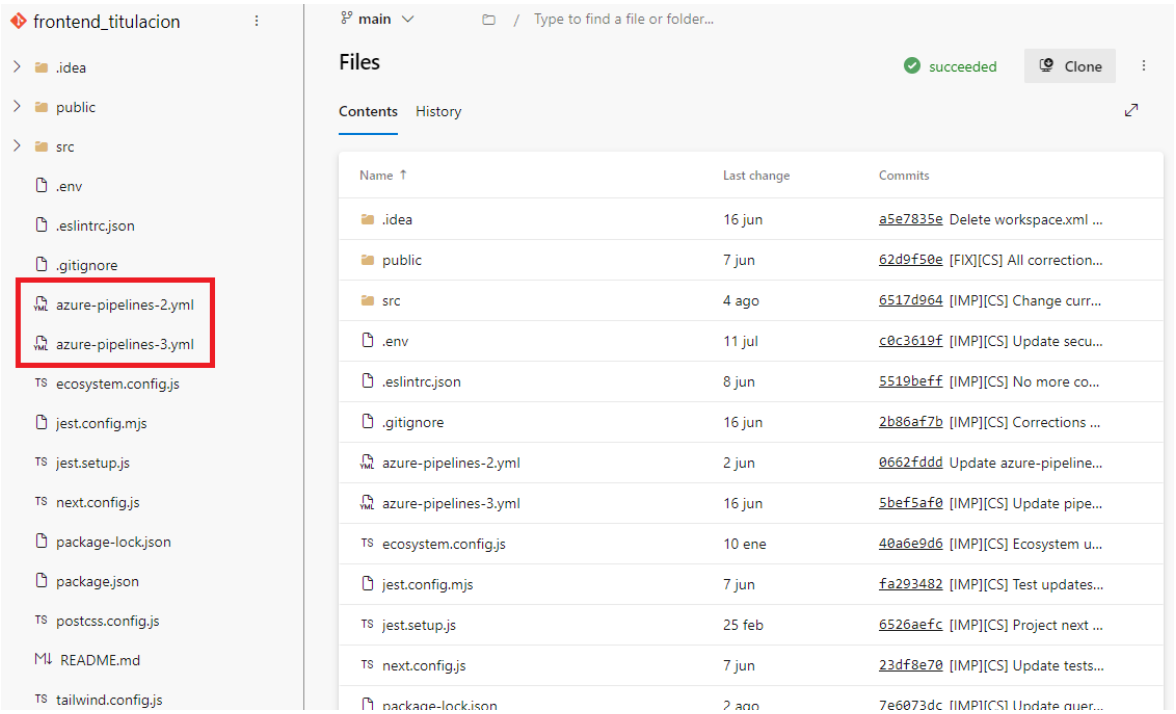


Figura 5. Repositorio para el Front End configurado en Azure DevOps

Configuración del Pipeline de CI para el Back End y el Front End

El proceso de configuración del pipeline, un flujo automatizado de trabajo que abarca todos los aspectos de la Integración Continua (CI), requirió una definición precisa de los procedimientos tanto para el Back End como para el Front End. En este contexto, se presentaron dos alternativas en la plataforma Azure DevOps: la creación visual del pipeline mediante una interfaz gráfica drag and drop; o la escritura directa en lenguaje YAML de las instrucciones de las tareas seleccionadas.

Configuración del Pipeline de CI para el Back End

La configuración del pipeline para el Back End tiene como objetivo principal automatizar y gestionar el proceso de construcción y entrega continua de los servicios para el Front End. Este pipeline se encuentra compuesto por varias etapas.

Es crucial establecer cuándo se activará el pipeline de construcción. El código fuente en formato YAML para activar el pipeline se presenta en la **Figura 6**. Este se configura para ejecutarse automáticamente ante cualquier cambio en la rama principal (main) del repositorio (línea 4). Además, se establece que afectará a todos los archivos y carpetas ubicados en la raíz del repositorio (línea 7). La opción "batch" se configura en "True" (línea 8), lo que implica la agrupación de cambios y la ejecución del pipeline en lotes.

```
1 trigger:
2   branches:
3     include:
4     - refs/heads/main
5   paths:
6     include:
7     - /
8   batch: True
```

Figura 6. Código fuente para activación del pipeline

En la **Figura 7** línea 1, se define el nombre del pipeline mediante una combinación de la fecha actual (con formato de año, mes y día) y un número de revisión. Esta estrategia asegura la identificación única de cada ejecución del pipeline.

```
1 name: $(date:yyyyMMdd)$(rev:.r)
```

Figura 7. Código fuente para la denominación del pipeline

La **Figura 8** marca el inicio de la definición de las tareas que constituirán el pipeline. Se establece un trabajo denominado, línea 2, "Phase_1" con la etiqueta visual "Agent job 1" (línea 3). El valor de "timeoutInMinutes" se fija en 0 (línea 4), lo que indica que no existe un límite temporal para la ejecución de este trabajo. Adicionalmente, se especifica que este trabajo se llevará a cabo en una máquina virtual con la imagen de Windows 2019 (línea 6).

```
1 jobs:
2 - job: Phase_1
3   displayName: Agent job 1
4   timeoutInMinutes: 0
5   pool:
6     vmImage: windows-2019
```

Figura 8. Código fuente para el inicio de la definición de tareas

Existe una secuencia de operaciones que componen el trabajo designado como "Phase_1". Este trabajo se caracteriza por la disposición secuencial de diversas etapas. A continuación, se desentrañarán cada una de las fases, destacando su función y aporte dentro del proceso de desarrollo.

- **Verificación del Código Fuente (Checkout: self):** En esta fase, se lleva a cabo la inspección del propio código fuente, permitiendo adquirir la versión más actualizada del repositorio.

- **Tarea – npm install:** Se ejecuta el comando "npm install" con el propósito de instalar las dependencias necesarias para el proyecto. Esta operación se configura de manera que no se muestren detalles en los mensajes (verbose: false).
- **Tarea – npm run test:** Mediante la ejecución del comando "npm run test" con una instrucción personalizada, se posibilita la realización de las pruebas del proyecto. De igual manera, se ajusta la configuración para que los mensajes detallados no sean mostrados (verbose: false).
- **Tarea - npm run test:e2e:** Se realiza la ejecución del comando "npm run test:e2e". Esta fase está destinada a las pruebas end-to-end (E2E) del proyecto. Asimismo, se establece que no se muestren detalles en los mensajes (verbose: false).
- **Tarea – npm run build:** Con la ejecución del comando "npm run build", se efectúa la construcción del proyecto. Esta operación se configura de tal manera que los mensajes detallados no sean mostrados (verbose: false).
- **Tarea - ArchiveFiles:** Este paso se encarga de crear un archivo comprimido (ZIP) que contendrá los archivos generados durante el proceso de construcción. Este archivo se ubica en una localización específica.
- **Tarea - PublishBuildArtifacts:** Por último, se procede a la publicación de los artefactos generados durante el proceso de construcción. En este caso, los archivos se publican bajo el nombre "drop".

En la **Figura 9**, se visualizan las diversas etapas que componen el pipeline y que fueron previamente mencionadas. Estas etapas abarcan desde la configuración del entorno y la ejecución de tareas de construcción (representadas en rojo), hasta la automatización de pruebas (en amarillo) y el empaquetamiento junto con la posterior publicación de artefactos (en verde). Esta metodología asegura una planificación detallada y coherente para garantizar la calidad, eficiencia y confiabilidad del proceso de desarrollo.

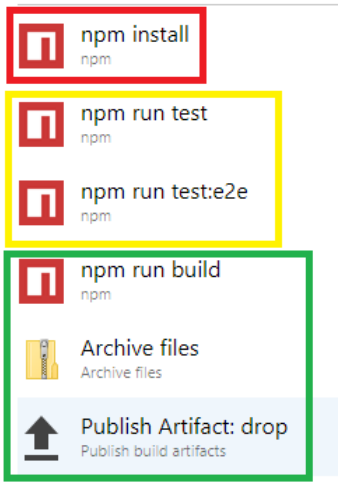


Figura 9. Pipeline visual para el Back End

En todas las tareas con un título que contiene la palabra "npm", se utiliza un comando personalizado que permite definir el comando npm que se desea ejecutar. En la **Figura 10**, se ilustra de manera gráfica cómo en la sección "command and arguments" es posible escribir el comando npm personalizado que se ejecutará en esta tarea del pipeline.

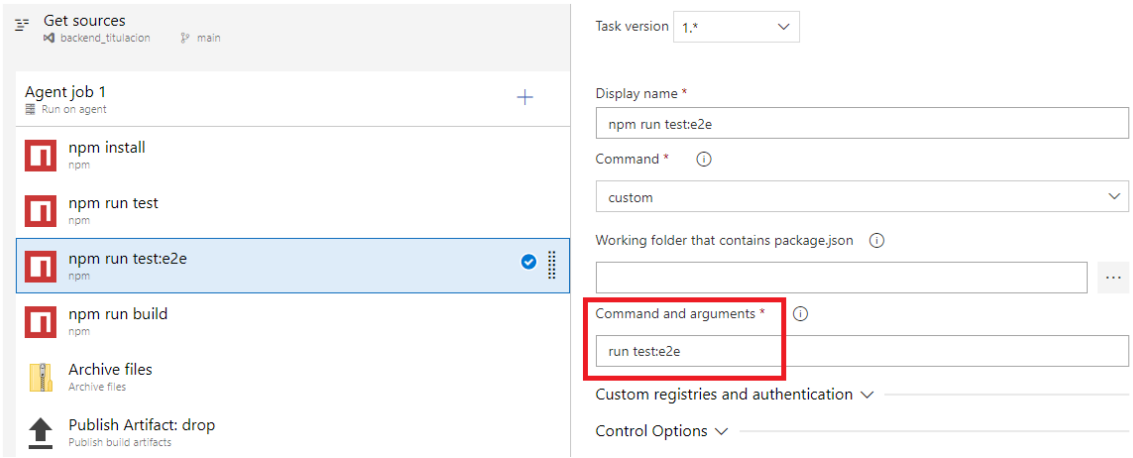


Figura 10. Interfaz de Azure DevOps donde se define un comando npm customizado

Dentro de la estructura del archivo YAML, la fase de empaquetamiento y publicación de artefactos se presenta como una secuencia de pasos cuidadosamente orquestados. Su finalidad es la de recolectar los componentes generados durante el proceso y prepararlos para su posterior utilización y distribución. Esta etapa garantiza la encapsulación y disponibilidad controlada de los resultados del proceso de construcción.

En esta fase, se ejecutan los siguientes pasos:

1. **Archive files:** Este paso realiza la tarea de empaquetar los archivos y directorios requeridos en un formato comprimido. Se configuran parámetros como el tipo de archivo de destino, el nivel de compresión y la ubicación del archivo empaquetado resultante. El proceso asegura la organización y eficiencia del empaquetado. En la **Figura 11**, se presenta esta configuración a través de la interfaz visual proporcionada por Azure DevOps.

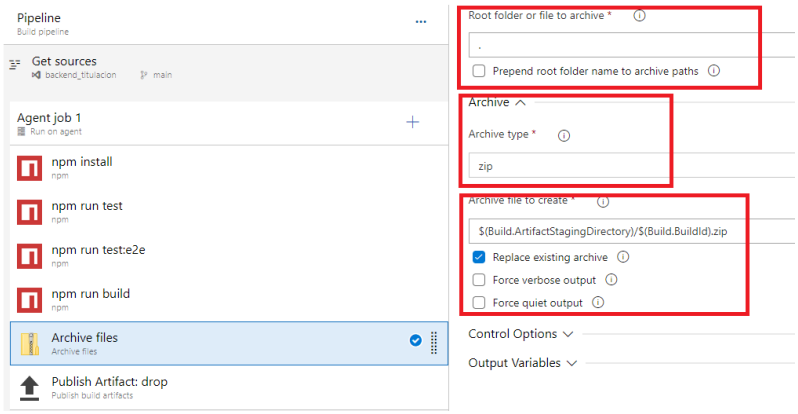


Figura 11. Interfaz de Azure DevOps para la tarea de empaquetamiento

2. **Publish Artifact: drop:** Posterior al empaquetamiento, este paso se encarga de la publicación de los artefactos empaquetados. Los artefactos, contenidos en el archivo comprimido, son transferidos y almacenados en un repositorio o ubicación designada. Se especifican detalles como la ruta del empaquetado, el nombre del artefacto y su tipo. En la **Figura 12**, se presenta esta configuración.

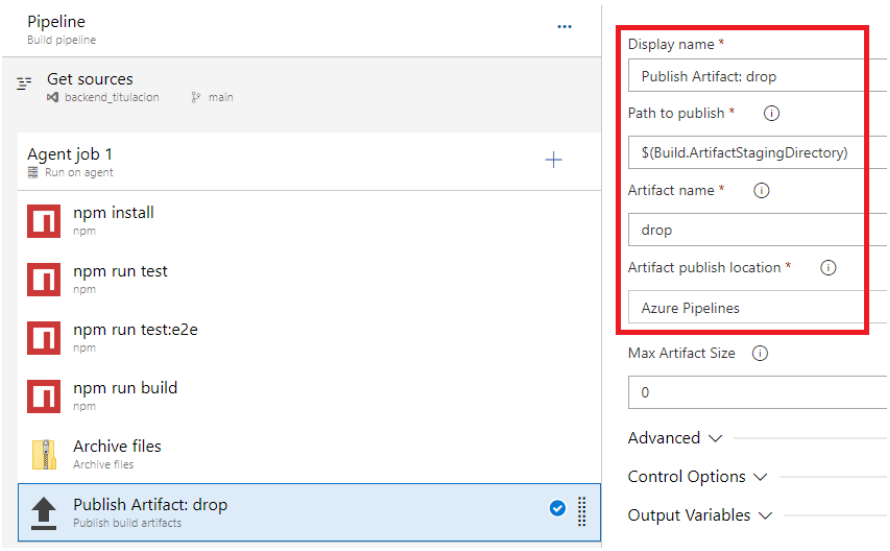


Figura 12. Interfaz de Azure DevOps para la publicación de artefactos

La fase de empaquetamiento y publicación garantiza que los componentes generados sean accesibles, controlados y listos para ser utilizados en etapas posteriores del ciclo de desarrollo. Cada paso de esta fase se encuentra configurado con condiciones para que solo se ejecute si las tareas anteriores han sido exitosas.

Este pipeline generado en Azure DevOps demuestra una planificación en la automatización de múltiples etapas esenciales para la gestión y despliegue del Back End. Cada fase se desarrolla asegura la calidad del código a través de pruebas, la generación de artefactos y la disponibilidad de una versión compilada.

Además, la integración con el sistema de control de versiones asegura la administración del código fuente a lo largo del ciclo de desarrollo y entrega. En conjunto, esta configuración del pipeline optimiza la eficiencia y confiabilidad del proceso de desarrollo de software.

Configuración del Pipeline de CI para el Front End

Para habilitar la integración continua del Front End, es esencial establecer un pipeline mediante el uso del lenguaje YAML. En la **Figura 13**, el inicio del pipeline se marca mediante un desencadenador (trigger) meticulosamente configurado para la rama principal (main) del repositorio visible en la línea 1 y 2. Asimismo, se delinea el entorno de ejecución al especificar una máquina virtual con el sistema operativo Ubuntu en su versión más reciente, visible en la línea 5.

```
1 trigger:
2 - main
3
4 pool:
5   vmImage: ubuntu-latest
```

Figura 13. Código fuente para desencadenar el pipeline de CI del Front End

Esta serie de pasos tiene un propósito específico y vital en el conjunto del flujo de trabajo. Estos procedimientos pueden apreciarse en la **Figura 14**, donde se exhibe el código fuente destinado a esta fase. El primer paso, por ejemplo, engloba la instalación de Node.js empleando la herramienta NodeTool (línea 2). Esta acción garantiza la presencia en la máquina virtual de la versión 18.x del entorno (línea 4).

```
steps:
- task: NodeTool@0
  inputs:
    versionSpec: '18.x'
  displayName: 'Install Node.js'
```

Figura 14.Código fuente para instalar Node.js

Posteriormente, se procede a ejecutar el script "npm install", que instala dependencias necesarias para el proyecto. Esta etapa asegura la disponibilidad de todas las librerías y módulos requeridos para llevar a cabo la construcción como se puede observar en la línea 2 de la Figura 15.

```
- script: |
    npm install
  displayName: 'npm install'
```

Figura 15. Código fuente para instalar Node.js

De manera adicional, en la **Figura 16** se introduce la implementación de una estrategia de caché para el directorio .next/cache (línea 2), la cual optimiza la velocidad y eficiencia del proceso de construcción, particularmente en las iteraciones subsiguientes.

```
- task: Cache@2
  displayName: 'Cache .next/cache'
  inputs:
    key: next | $(Agent.OS) | package-lock.json
    restoreKeys: |
      npm | $(Agent.OS)
    path: '$(System.DefaultWorkingDirectory)/.next/cache'
```

Figura 16. Código fuente para controlar el caché

El proceso continúa con el comando "npm run build", responsable de llevar a cabo la construcción del proyecto. Esta fase implica la compilación y generación de los archivos y componentes necesarios para el funcionamiento de la aplicación. Estos detalles pueden observarse en la **Figura 17**.

```
1 - script: |
2     npm run build
3   displayName: 'npm build'
```

Figura 17. Código fuente para la construcción del proyecto

En la **Figura 18** se muestra el código fuente para la creación de un archivo comprimido (ZIP) que contiene los resultados de la construcción. Esta tarea se lleva a cabo mediante el uso del módulo "ArchiveFiles" (línea 2). El archivo resultante se nombra de acuerdo con el ID de la compilación (línea 7) y se almacena en el directorio de artefactos de la compilación, de igual forma en la línea 7. Por último, en la línea 8 se define que se sobrescriba si existe un elemento con ese nombre.

```
1 - task: ArchiveFiles@2
2   displayName: 'Archive files'
3   inputs:
4     rootFolderOrFile: '$(System.DefaultWorkingDirectory)'
5     includeRootFolder: false
6     archiveType: 'zip'
7     archiveFile: '$(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip'
8     replaceExistingArchive: true
```

Figura 18. Código fuente para la creación del archivo comprimido

Finalmente, en la **Figura 19**, se muestra la configuración se efectúa la publicación de los artefactos generados utilizando la tarea "PublishBuildArtifacts" (línea 1). Los archivos empaquetados son transferidos y almacenados en un repositorio (línea 3) o ubicación específica bajo el nombre "drop" (línea 4). Esta acción permite que los componentes construidos estén disponibles para su uso y despliegue posterior. previamente mencionada.

```
1 - task: PublishBuildArtifacts@1
2   inputs:
3     PathtoPublish: '$(Build.ArtifactStagingDirectory)'
4     ArtifactName: 'drop'
5     publishLocation: 'Container'
```

Figura 19. Código fuente para la publicación del archivo comprimido

En síntesis, esta estructura del pipeline del Front End en formato YAML constituye una estrategia global que promueve una construcción y despliegue eficaz de la aplicación. Cada fase desempeña un rol importante en la gestión de dependencias, el proceso de edificación y la formación de paquetes, colaborando en la formación

de un procedimiento fluido y confiable. La inclusión de herramientas de caché y la optimización de recursos se fusionan para asegurar la eficiencia y calidad en la trayectoria de desarrollo y despliegue del software.

Configuración del Pipeline de CD para el Back End y el Front End

Esta sección se centra en exponer los procedimientos empleados para establecer un flujo de trabajo para optimizar la entrega de software en ciclos de desarrollo y despliegue.

En la implementación exitosa del CI/CD, se examinaron las fases y decisiones involucradas en la configuración del pipeline de CD para ambos componentes del sistema. Este apartado permite comprender cómo la configuración de un pipeline impulsa la eficiencia y la calidad en el proceso de desarrollo y entrega, alineándose con los objetivos del sistema.

Configuración del Pipeline de CD para el Back End

El proceso de configuración del pipeline destinado al despliegue y entrega continua del Back End se llevó a cabo aprovechando las funcionalidades del asistente visual proporcionado por Azure DevOps.

Es fundamental destacar que este proceso se inicia con el artefacto, el cual se origina como resultado del pipeline de integración continua. Para esta etapa, se empleó la interfaz gráfica que ofrece la herramienta, donde se detalló el origen del artefacto proveniente del pipeline de integración.

La **Figura 20** ofrece una representación visual del pipeline de despliegue, donde se resalta en tono rojo la selección del pipeline del cual se extraerá el artefacto resultante del pipeline de integración continua.

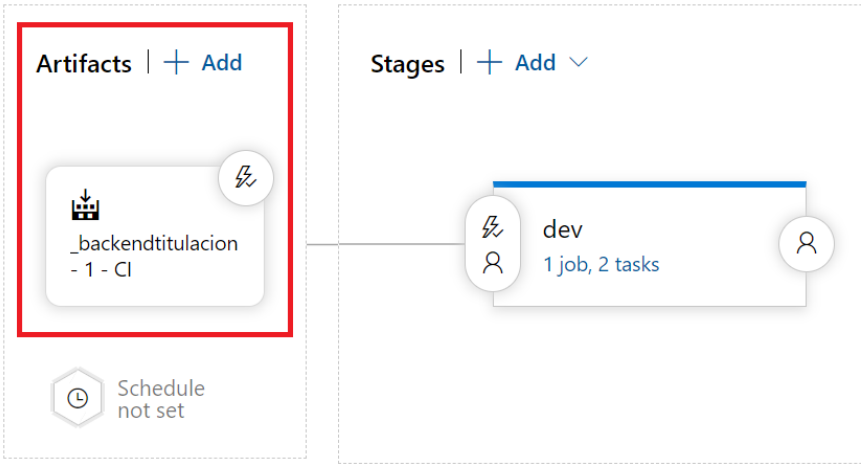


Figura 20. Representación visual del pipeline de despliegue del Back End

El pipeline de despliegue depende de la tecnología de la aplicación, específicamente en el caso del Back End, donde se hace uso de Node.js.

Simultáneamente, se procedió a definir la ubicación del paquete de la aplicación destinado al despliegue, comúnmente contenido en un archivo comprimido con formato ZIP. Además, se especificó el comando que instruye la manera en que la aplicación debe iniciar en el servidor web.

Estos parámetros cruciales se presentan visualmente en la **Figura 21**, siendo destacado en rojo el tipo de tecnología empleado, en amarillo la ubicación del paquete de la aplicación y en verde el comando de inicio correspondiente.

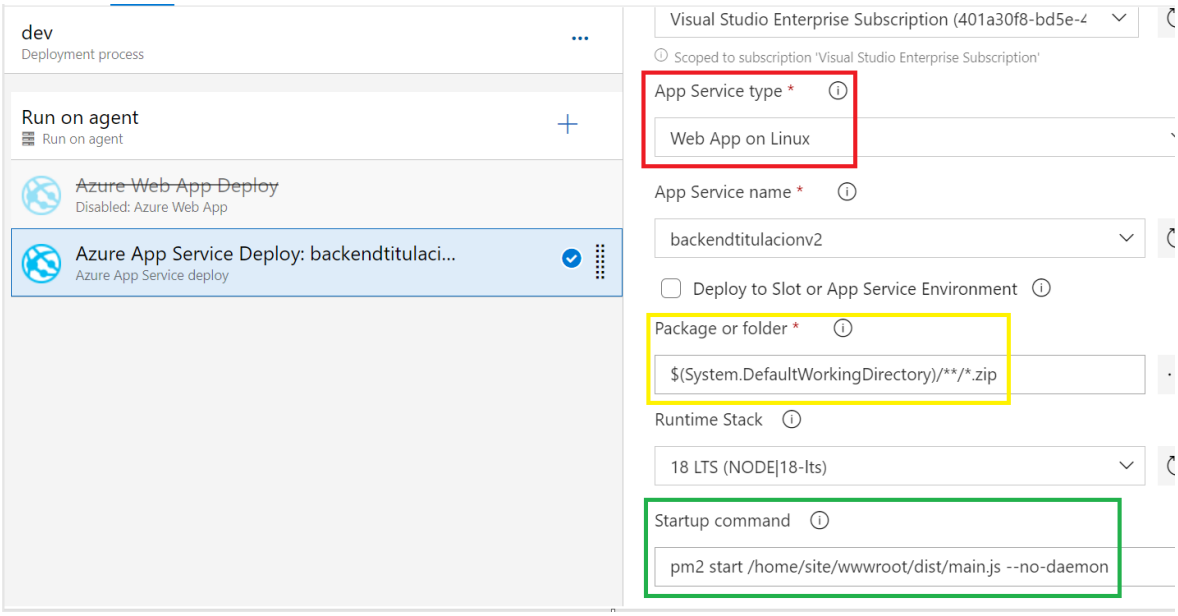


Figura 21. Interfaz para configuración del pipeline del Back End

Configuración del Pipeline de CD para el Front End

Existe otra alternativa para la configuración del pipeline que difiere del asistente visual de Azure DevOps: la posibilidad de establecer el pipeline utilizando lenguaje YAML. En el contexto del despliegue del Front End, la configuración del pipeline adquiere un papel de suma importancia debido a las tareas esenciales que engloba. La ejemplifica el archivo en formato YAML que define el proceso de despliegue. En la **Figura 22** se evidencia la ejecución del despliegue se logra a través de la tarea "AzureRmWebAppDeployment@4" (línea 2), encargada de gestionar la configuración y ejecución del proceso de despliegue.

```
1  steps:
2  - task: AzureRmWebAppDeployment@4
3    displayName: 'Deploy Azure App Service'
4    inputs:
5      azureSubscription: '$(Parameters.ConnectedServiceName)'
6      appType: '$(Parameters.WebAppKind)'
7      WebAppName: '$(Parameters.WebAppName)'
8      packageForLinux: '$(System.DefaultWorkingDirectory)/_frontend_titulacion v2/drop/*.zip'
9      RuntimeStack: 'NODE|18-lts'
10     StartupCommand: '$(Parameters.StartupCommand)'
```

Figura 22. Código fuente del pipeline del despliegue del Front End

Dentro de esta tarea, los parámetros se detallan mediante la sección "inputs" (línea 4 a la 10). En primer lugar, se suministra la información relativa a la suscripción de Azure destinada al despliegue. El valor del parámetro "azureSubscription" se determina mediante la variable "$(Parameters.ConnectedServiceName)" (línea 5), la cual incorpora el nombre de la suscripción correspondiente al sistema.

El siguiente parámetro relevante es "appType", el cual establece la naturaleza de la aplicación que se despliega en Azure. Dicho valor es extraído de la variable "$(Parameters.WebAppKind)", portadora de la información acerca de la tecnología de la aplicación, en este caso, Node.js, configuración visible en la línea 6.

La sección "WebAppName" se emplea para identificar el nombre de la aplicación web en Azure donde se llevará a cabo el despliegue, visible en la línea 7. Este nombre proviene de la variable "$(Parameters.WebAppName)".

La propiedad "packageForLinux", línea 8, alude a la ubicación del paquete de la aplicación a desplegar, usualmente un archivo comprimido en formato ZIP que contiene los recursos necesarios para la aplicación del Front End. La ruta del paquete se construye utilizando "$(System.DefaultWorkingDirectory)", visible en la línea 8, como referencia al directorio de trabajo por defecto y se concatena con la ruta relativa al paquete.

Además, se especifica la pila de tiempo de ejecución en la que la aplicación operará en Azure. En este caso, se emplea la versión 18-lts de Node.js, establecida a través de la propiedad "RuntimeStack", visible en la línea 9.

Finalmente, el comando de inicio de la aplicación se introduce mediante la variable "$(Parameters.StartupCommand)", visible en la línea 10, habitualmente requerida para indicar la forma en que la aplicación debe arrancar en el servidor web.

En síntesis, esta sección del pipeline configura y ejecuta la entrega de una aplicación en el servicio de aplicaciones de Azure. Los parámetros suministrados permiten personalizar el despliegue conforme a la suscripción, tipo de aplicación, nombre de la aplicación web, ubicación del paquete, pila de tiempo de ejecución y comando de inicio específicos.

Implementación de Pruebas

Dentro del contexto de los pipelines de integración y despliegue, la implementación de pruebas es primordial en la búsqueda de la excelencia en el desarrollo de software. Este apartado se centra en la planificación y ejecución de tres tipos de pruebas esenciales: pruebas unitarias, pruebas end-to-end y pruebas de carga.

Se integraron pruebas de manera coherente en el flujo de trabajo de los pipelines, las cuales juegan un papel clave en la validación y verificación del código en diferentes niveles. Las pruebas unitarias analizan las unidades individuales de código, mientras que las pruebas end-to-end evalúan la interacción entre componentes. Además, las pruebas de carga permiten evaluar la resistencia y capacidad de respuesta del sistema ante condiciones simuladas de alta demanda.

En conjunto, estos enfoques de prueba contribuyeron a fortalecer la calidad y confiabilidad del software, garantizando una entrega exitosa y una experiencia fluida para los usuarios finales.

Pruebas Unitarias

En el contexto del servicio de respuestas en una aplicación desarrollada con el marco de trabajo NestJS, se han implementado pruebas que se enfocan en el Back End. El objetivo de estas pruebas es asegurar el funcionamiento sin fallos de los componentes del servicio y su interacción con otros servicios y repositorios.

La suite de pruebas creada para los servicios del Back End se presenta como una estrategia clave para verificar tanto la funcionalidad como el comportamiento del servicio de respuestas en un entorno controlado, como se puede observar en la **Figura 23**. Cada prueba individual se orienta hacia una parte específica de los servicios expuestos por el Back End, lo que asegura la correcta definición y el funcionamiento correcto de estos componentes, así como su adecuada interacción con otros elementos del sistema. En la Figura 23 se puede observar los nombres de las pruebas que conforman la suite.

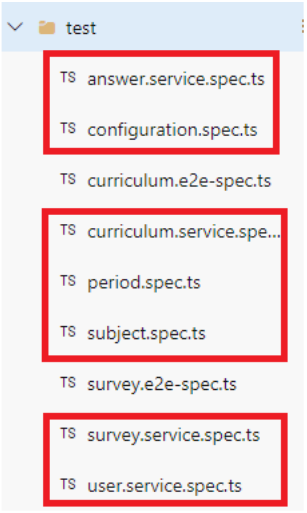


Figura 23. Suite de pruebas unitarias del Back End

Las pruebas unitarias son ejecutadas automáticamente al detectarse cambios en la rama principal. Así, cada ejecución del pipeline de integración incluyó la ejecución y presentación de resultados de las pruebas unitarias. En la **Figura 24** línea 44, podemos ver los resultados de las pruebas unitarias.

```
43  Test Suites: 8 passed, 8 total
44  Tests:       28 passed, 28 total
45  Snapshots:   0 total
46  Time:        17.737 s
47  Ran all test suites.
48  > backend_titulacion@0.0.1 test
49  > jest
```

Figura 24. Resultados de las pruebas unitarias del Back End

En el contexto del Front End, se han implementado pruebas utilizando el framework de pruebas Jest para verificar la correcta integración y funcionamiento de los servicios previamente expuestos y probados en el Back End. Estas pruebas tienen como objetivo principal validar si las solicitudes de consumo de los servicios desde el Front End retornan las respuestas esperadas. La suite de pruebas del Front End presenta múltiples casos de prueba diseñados para abordar diferentes aspectos de la funcionalidad de los servicios relacionados con el sistema.

Cada caso de prueba sigue una estructura, donde se definen los parámetros para las solicitudes, se realiza la invocación de las funciones que gestionan las peticiones HTTP y finalmente se evalúan las respuestas obtenidas. Estos casos de prueba destacan por utilizar diversas combinaciones de valores simulados para los parámetros, con el objetivo de examinar el comportamiento de las funciones ante diferentes condiciones. Esta estrategia de pruebas del Front End busca reforzar la calidad y robustez del sistema, asegurando su adecuada interacción con el Back End y la entrega de resultados confiables y esperados en diversos escenarios como se observa en la Figura 25.

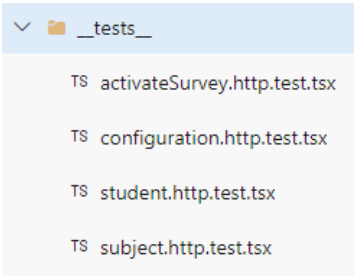


Figura 25. Suite de pruebas unitarias del Front End

Las pruebas unitarias son ejecutadas automáticamente al detectarse cambios en la rama principal o en la rama de desarrollo. Así, cada ejecución del pipeline de integración incluyó la ejecución y presentación de resultados de las pruebas unitarias. En la **Figura 26** línea 27, podemos ver los resultados de las pruebas unitarias.

```
24  Test Suites: 4 passed, 4 total
25  Tests:       5 passed, 5 total
26  Snapshots:   0 total
27  Time:        3.235 s
```

Figura 26. Resultados de las pruebas unitarias del Front End

Pruebas End-to-End

Dentro del ámbito de las pruebas end-to-end en el Back End, se ha implementado una suite de pruebas que abordan diferentes aspectos funcionales. Esta suite de pruebas se enfoca en garantizar la correcta interacción y funcionamiento del módulo, así como su integración con otros componentes del sistema. En el contexto de estas pruebas, se ha utilizado el módulo supertest junto con el enfoque de solicitud y respuesta HTTP, lo que permite simular interacciones con el servidor real.

Para lograrlo, se configuró un entorno controlado donde se inicializa la aplicación utilizando. Este entorno establece una serie de pruebas unitarias que se ejecutan en secuencia para validar la funcionalidad de diferentes rutas del módulo de currículo.

En la **Figura 27** se define la ubicación de los archivos de pruebas en la carpeta "../src" (línea 3), establece el entorno de prueba como "node" (línea 4), y selecciona los archivos de especificación de pruebas que tienen la extensión ".e2e-spec.ts", visible en la línea 5. Además, utiliza "ts-jest" como transformador para manejar archivos TypeScript y JavaScript, como se puede observar en la línea 7.

```
1 {
2   "moduleFileExtensions": ["js", "json", "ts"],
3   "rootDir": "../src",
4   "testEnvironment": "node",
5   "testRegex": ".e2e-spec.ts$",
6   "transform": {
7     "^.+\\.(t|j)s$": "ts-jest"
8   }
9 }
```

Figura 27. Configuración del entorno para las pruebas end-to-end

Cada caso de prueba evalúa una ruta específica utilizando la función request de supertest y verifica la respuesta obtenida en términos de código de estado y contenido del cuerpo de la respuesta. Por ejemplo, en la **Figura 28** en la línea 5 se muestra esta configuración. Esta estructura repetitiva de pruebas permite asegurar la coherencia y consistencia de las funcionalidades expuestas por el módulo, lo que contribuye a la fiabilidad y robustez del sistema en su conjunto.

```
1 it('get-all-periods', async () => {
2     const response = await request(app.getHttpServer())
3       .get('/curriculum/get-all-periods')
4       .set({ Authorization: token });
5     expect(response.statusCode).toBe(200);
6     expect(response.body).toBeDefined();
7     return response;
8   });
```

Figura 28. Ejemplo de prueba unitaria para la prueba end-to-end

En resumen, esta suite de pruebas end-to-end en el Back End representa un enfoque integral para garantizar la correcta operación y la integración del módulo de currículo en la aplicación. Mediante una serie de pruebas unitarias bien definidas, se evalúa de manera minuciosa cada una de las rutas relevantes, asegurando su comportamiento en términos de interacción HTTP y el manejo adecuado de las respuestas. Esto fortalece la confiabilidad y calidad del sistema, permitiendo identificar y abordar posibles problemas en una etapa temprana del proceso de desarrollo. Los resultados se pueden observar en la **Figura 29**.

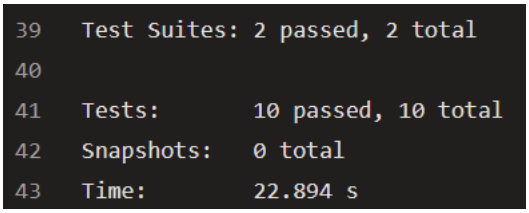


Figura 29. Resultados de las pruebas ent-to-end

Pruebas de Carga

Este apartado se enfoca en la ejecución de pruebas de carga en el Front End de la aplicación, utilizando la herramienta LoadRunner de Micro Focus. Estas pruebas tienen como propósito comprender la respuesta y el rendimiento de la aplicación ante cargas significativas, brindando información para tomar decisiones con relación a la escalabilidad y ajuste del sistema.

A continuación, se detalla el proceso de ejecución de las pruebas de carga, describiendo los escenarios diseñados, los protocolos de comunicación utilizados y los parámetros de configuración específicos.

Estas pruebas evaluaron la capacidad del Front End para enfrentar situaciones de alta demanda. En última instancia, este enfoque integral contribuye a identificar y mitigar posibles problemas de rendimiento antes de que afecten la experiencia del usuario final.

El enfoque en la realización de pruebas de carga mediante LoadRunner se basa en la necesidad de simular situaciones de uso realistas para determinar cómo el Front End interactúa con el servidor bajo alta demanda. LoadRunner, una herramienta ampliamente reconocida en la industria permite recrear escenarios complejos, evaluar tiempos de respuesta y tasas de éxito, y detectar posibles cuellos de botella en la arquitectura del sistema.

Estas pruebas buscan obtener una comprensión profunda del rendimiento del Front End en términos de latencia, capacidad y estabilidad en situaciones de carga intensificada, con el objetivo de garantizar una experiencia de usuario óptima incluso durante momentos de alta demanda.

La generación de la prueba implicó la definición de características esenciales para simular una experiencia de uso realista, en la que un total de 30 usuarios actuarían de manera concurrente. Estos usuarios, con el rol de estudiantes, procederían a iniciar sesión en el aplicativo web durante un período de 5 minutos, reflejando el tiempo promedio estimado para completar la encuesta y luego cerrar sesión.

Este escenario se planificó en base a métricas y la comprensión de los patrones de uso típicos de los usuarios objetivo: estudiantes de las carreras de Ingeniería de Software e Ingeniería de Computación. La duración de 5 minutos para la estabilidad del escenario se determinó considerando el tiempo medio que se tardaría en finalizar la encuesta de preplanificación.

La elección del perfil de usuario se enfocó en los estudiantes, debido a su potencial para generar una concurrencia significativa que pudiera afectar el rendimiento de la aplicación. La Tabla 3. Transacciones definidas para el escenario detalla las transacciones definidas para este escenario, junto con su respectiva explicación, para un mejor entendimiento del flujo de la prueba.

Tabla 3. Transacciones definidas para el escenario

Código de la Transacción	Descripción
Trx01URL	Ingresar la url del aplicativo web
Trx02Login	Proceso de login en el aplicativo web
Trx03EnviarMaterias	Enviar materias seleccionadas
Trx04Logout	Cerrar sesión en el aplicativo web

En la herramienta LoadRunner, se desarrolló el script de ejecución de la prueba de manera intuitiva, incorporando los parámetros que los usuarios virtuales, conocidos como "vusers" en este contexto, utilizarían para llevar a cabo un ingreso exitoso y una simulación efectiva del envío de la encuesta. La **Figura 30** muestra la interfaz de LoadRunner utilizada para configurar las transacciones y los campos requeridos en el proceso.

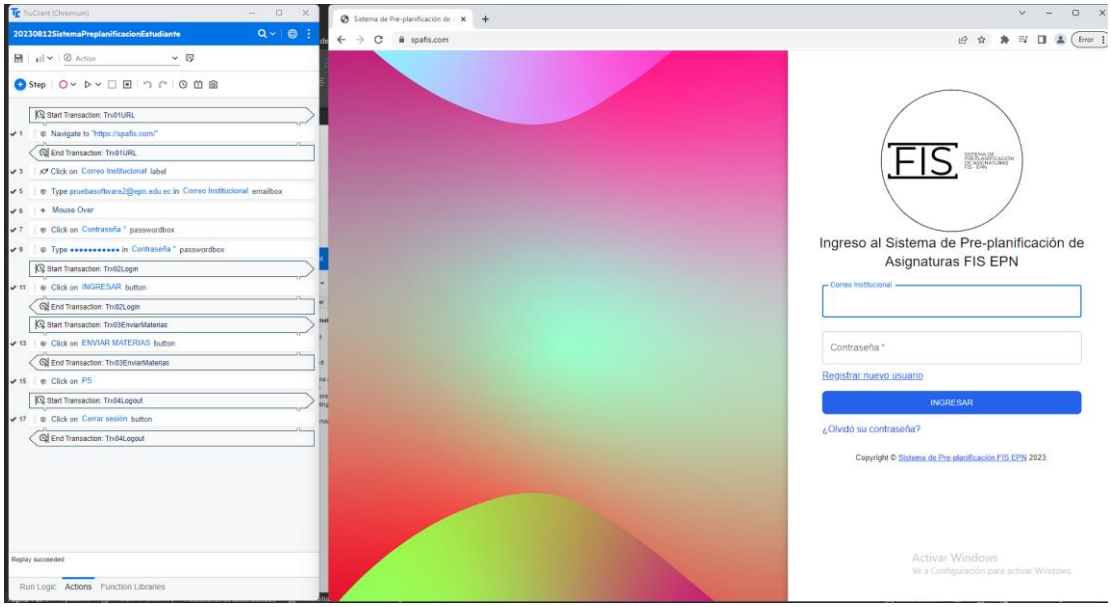


Figura 30. Interfaz de LoadRunner para la configuración de la prueba

El escenario de carga implementado simuló la progresiva conexión de 5 usuarios cada dos minutos, hasta alcanzar la simultaneidad de 30 usuarios. Este enfoque se presenta visualmente en la **Figura 31**, demostrando el aumento gradual en el número de usuarios virtuales a lo largo del tiempo.

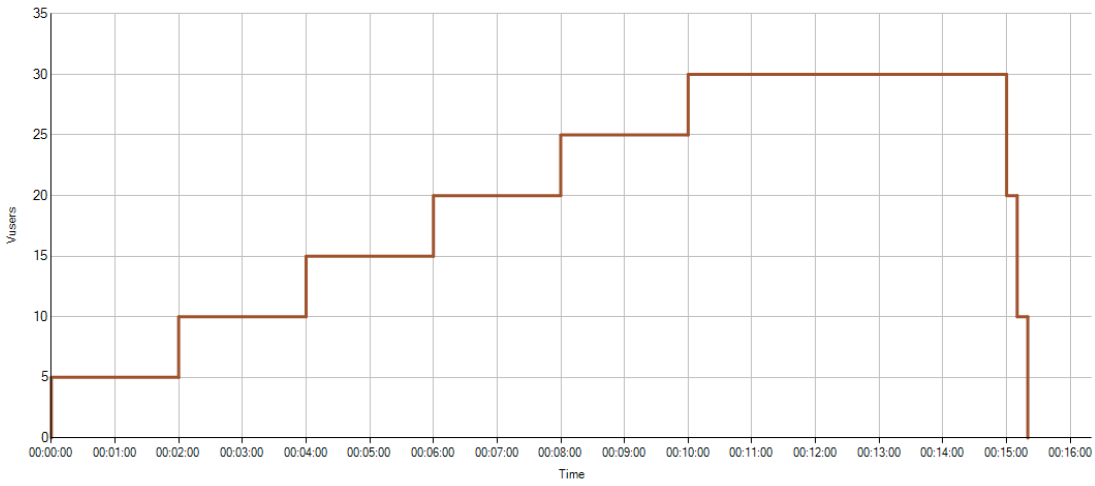


Figura 31. Gráfico de usuarios virtuales con relación al tiempo

Al culminar la implementación y configuración de los pipelines de integración y despliegue, junto con todas las etapas que los engloban, se logra concretar un sistema holístico y perfectamente automatizado. Este entramado de procesos proporciona no solo una estructura sólida para el desarrollo continuo y la entrega efectiva de software, sino también una base para asegurar la calidad y el rendimiento constante de la aplicación.

Soporte de Azure DevOps a Scrum

Azure DevOps, mediante su conjunto de herramientas como Boards, respaldó eficazmente la implementación del marco Scrum. Esto posibilitó la definición, estimación y seguimiento de historias de usuario y tareas correspondientes a cada sprint. En la Tabla 4 se presentan los objetivos establecidos por sprint, y en la **Figura 32** se ilustra cómo Boards proporcionó al equipo una visión del estado y progreso de las tareas a lo largo del proceso.

Tabla 4. Objetivos del sistema planteados por sprint

Sprint	Azure DevOps
1	Elaborar formularios por carrera y malla curricular.
2	Personalizar formularios por estudiante
3	Elaborar formularios de administración
4	Generar malla curricular por PAO
5	Corregir errores y mejorar experiencia de usuario
6	Carga de curriculum académicos de estudiantes
7	Generar reportes personalizados

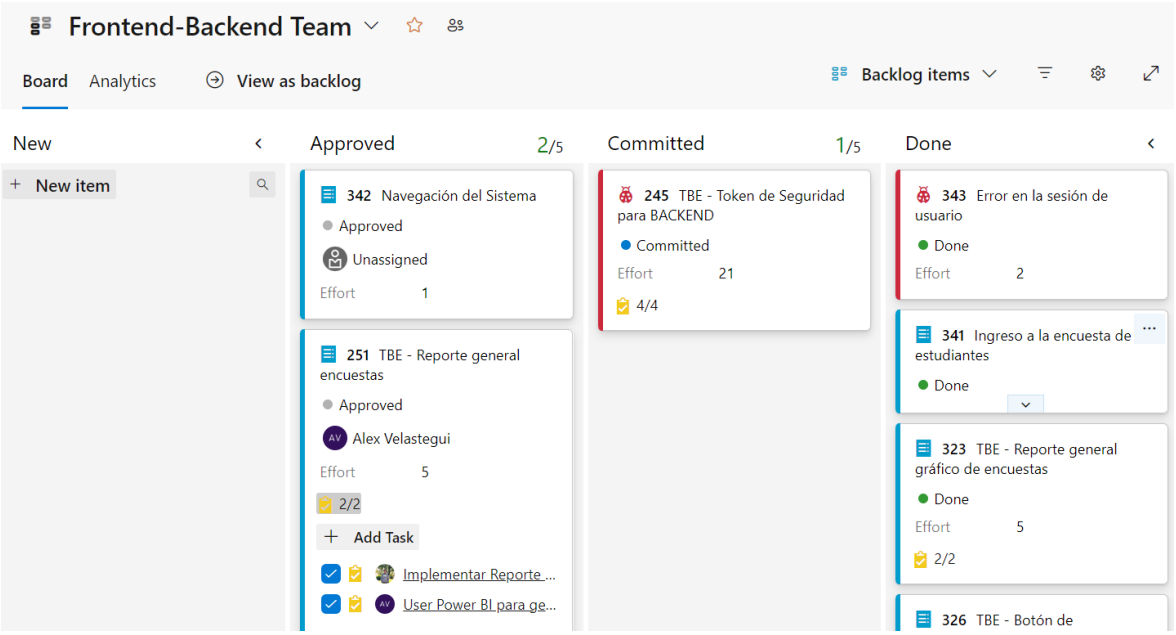


Figura 32. Interfaz de Boards de Azure DevOps

Azure DevOps, con su conjunto de herramientas, desempeñó un papel fundamental al centralizar los componentes del sistema en una única plataforma, permitiendo la generación de incrementos funcionales que alcanzaron los objetivos establecidos por sprint en el proyecto. La Figura 33 presenta la interfaz principal del sistema, donde se visualizan los módulos que satisfacen las necesidades del proyecto de manera efectiva.

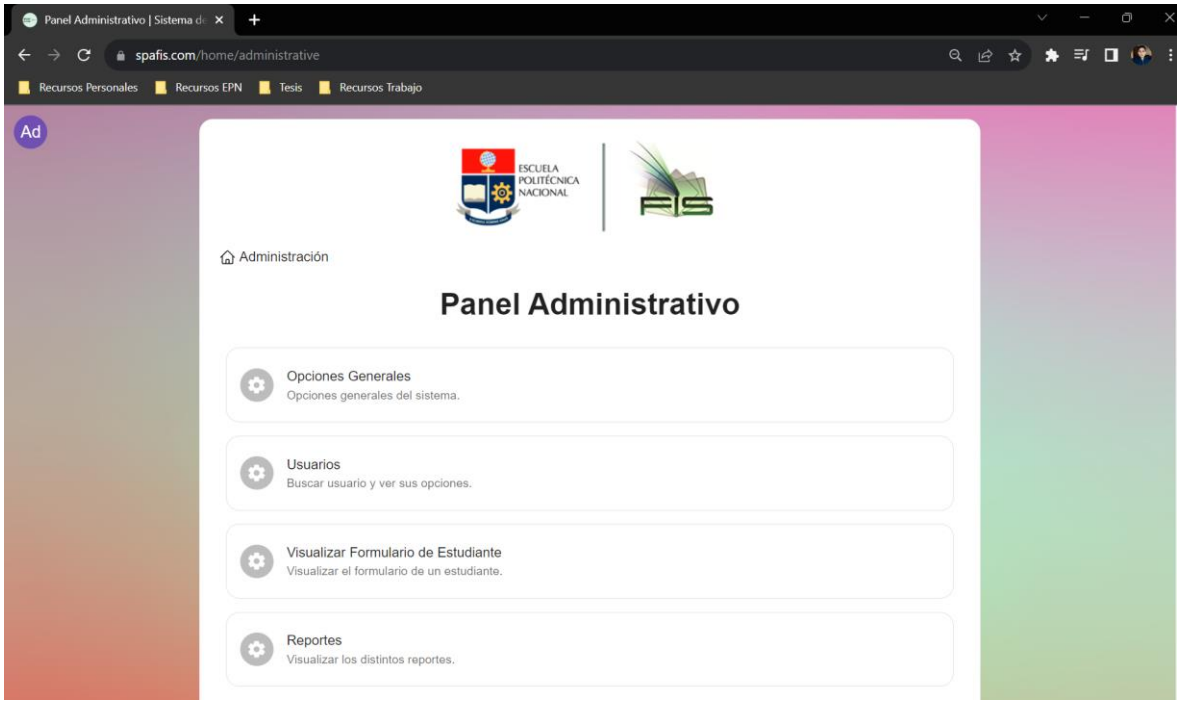


Figura 33. Interfaz de módulos del sistema de preplanificación para la FIS

En el ANEXO I se encuentra una demostración y descripción más amplia del sistema.

En la siguiente sección, se presentarán y analizarán en detalle los resultados obtenidos de la ejecución de las pruebas y evaluaciones correspondientes, seguido de un análisis de conclusiones y recomendaciones que derivan de la experiencia y los hallazgos obtenidos. La conexión de las diferentes fases refuerza la contribución de la automatización y la integración en la construcción de sistemas eficientes y confiables en un entorno de desarrollo ágil y competitivo.

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

3.1 Resultados

Una vez implementados satisfactoriamente todos los componentes del sistema de preplanificación de materias para la FIS, es posible analizar y conferir significado a las salidas de los diversos procesos. La elección de la herramienta Azure DevOps centralizó esta información en artefactos importantes como los repositorios del Back End y Front End, Pipelines de integración y despliegue, Wiki, Product y Sprint Backlogs. Brindando respaldo tanto a la integración continua de los componentes como a la gestión del producto, al tiempo que respaldaba el marco Scrum.

La importancia de que cada componente funcione individualmente y que los componentes trabajen en conjunto se tradujo en el logro de los objetivos establecidos por cada sprint, los cuales fueron planificados en base a los objetivos acordados por los miembros del equipo de desarrollo y el Product Owner. Esto aseguró que la creación y cualquier modificación de los pipelines de integración y despliegue no impactaran significativamente en los demás componentes.

Azure DevOps ayudó a que todas las etapas de construcción del sistema operaran de manera transparente para los miembros del equipo Scrum. Las modificaciones a dichas etapas no afectaron el avance individual de los demás componentes, sino que proporcionaron soporte automatizado para tareas repetitivas, permitiendo que el enfoque se mantuviera en la adición de funcionalidad, más que en la forma de realizar dicha entrega.

El servidor de integración continua de Azure DevOps logró que cada cambio detectado en el sistema de control de versiones automáticamente inicie las tareas necesarias para integrar y desplegar el producto. La **Figura 34** muestra los resultados de cada uno de los pipelines, ofreciendo trazabilidad en cada etapa del ciclo CI/CD y resaltando errores si existen.

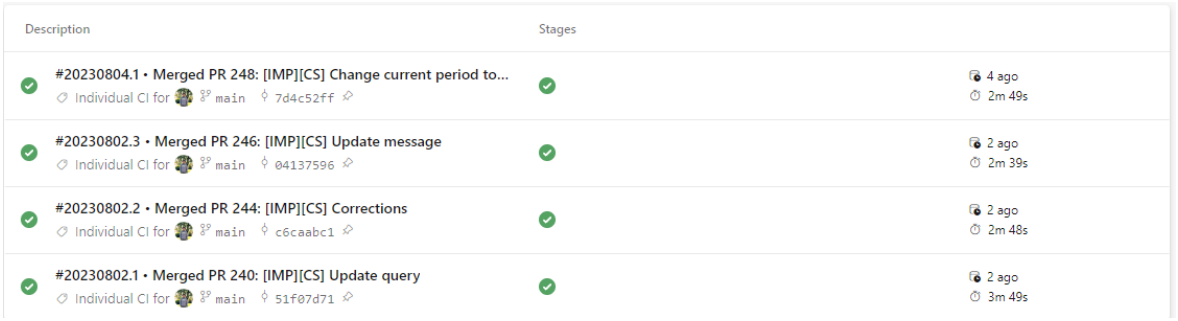


Figura 34. Resultados de las ejecuciones del pipeline general de Front End

En la **Figura 35** se presenta una descripción de cada etapa considerada en el pipeline, junto con su resultado, representado en verde para éxito o en rojo si se detecta un error. Los errores se detallan en la consola, asegurando la trazabilidad del error y su ubicación. Adicionalmente se presenta el tiempo de ejecución del pipeline el cuál es un dato importante.

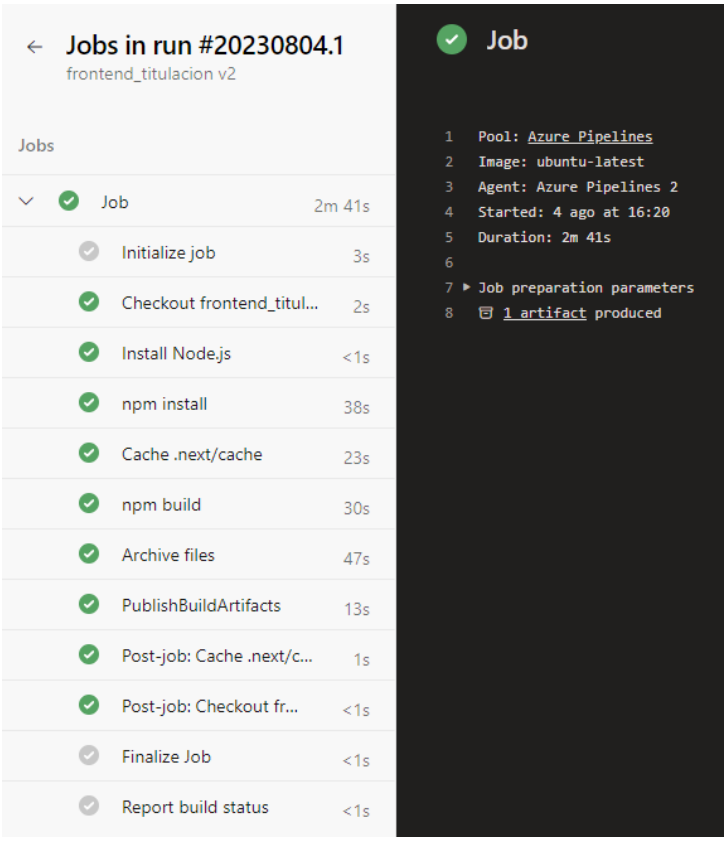


Figura 35. Ejecución etapa por etapa del pipeline general del Front End

Una ventaja importante es que un único disparador (trigger) permite el inicio de todas las etapas del ciclo CI/CD. Si alguna etapa falla, el pipeline se detiene antes de las etapas del CD, permitiendo la corrección y repetición del proceso mediante la activación transparente de un desencadenador para el equipo de desarrollo.

Además, Azure DevOps ofrece métricas que permiten conocer la cantidad de ejecuciones del pipeline y su tasa de éxito. Estos valores cobran relevancia al analizar con mayor detalle el rendimiento del equipo de trabajo. La **Figura 36** muestra el gráfico con el porcentaje de ejecuciones exitosas del pipeline de CI del Front End, basado en datos de los últimos 180 días.

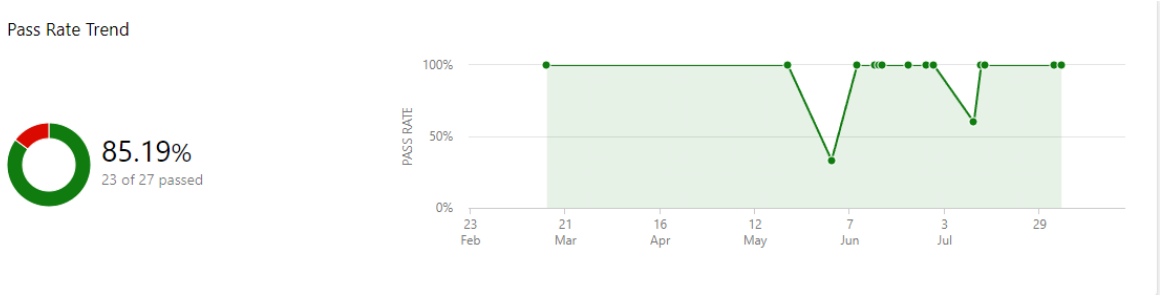


Figura 36. Gráfico con porcentaje de ejecuciones del pipeline general del Front End

Una de las etapas que se consideró más importante al momento de definir el pipeline de CI/CD es tener una suite de pruebas automatizadas para tener cobertura en el código, Azure DevOps nos permite revisar el número de pruebas ejecutadas que pasaron satisfactoriamente, que fallaron y el tiempo que tomó el ejecutarlas. Permitiendo al equipo de desarrollo crear suite de pruebas, que están siendo automatizadas gracias al servidor de integración. En la **Figura 37** se visualiza como se ejecutó la suite de pruebas unitarias y por los resultados que se muestran por consola.

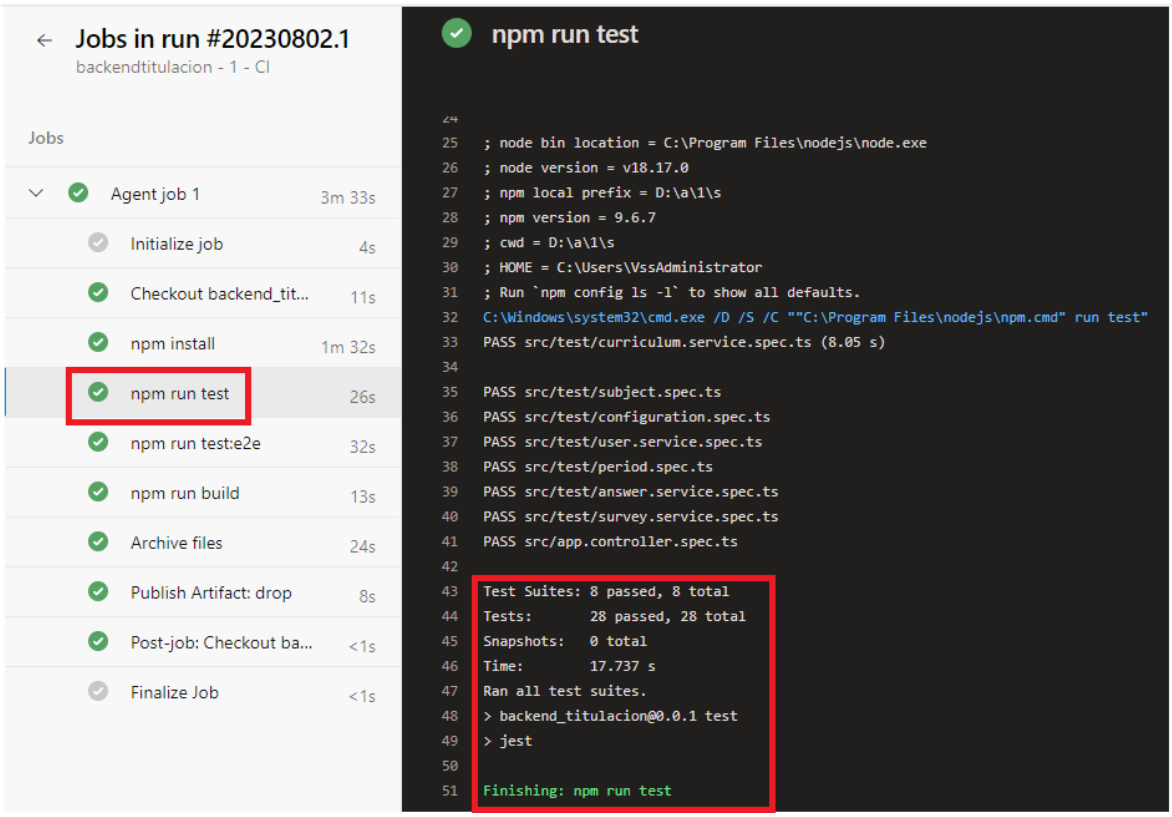


Figura 37. Ejemplo de ejecución de suite pruebas en el pipeline general del Back End

Es fundamental abordar los resultados de las pruebas, incluyendo la prueba de carga implementada. Esto brinda la oportunidad de verificar cómo la conexión entre las diversas fases de integración y despliegue continuo refuerza la contribución de la automatización y la integración en la construcción de sistemas eficientes y confiables en un entorno de desarrollo ágil y competitivo.

El escenario considerado involucra a un número de estudiantes que acceden de manera concurrente para completar la encuesta del sistema de preplanificación. Este escenario abarca el flujo ideal, donde los usuarios ingresan sus credenciales correctamente, completan la encuesta en aproximadamente 5 minutos y finalizan el proceso.

Los resultados de la prueba, que comprendió un total de 1138 transacciones realizadas por 30 usuarios virtuales, indicaron una duración de 16 minutos y 16 segundos. La **Figura 38** muestra el recuento de transacciones por cada actividad definida.

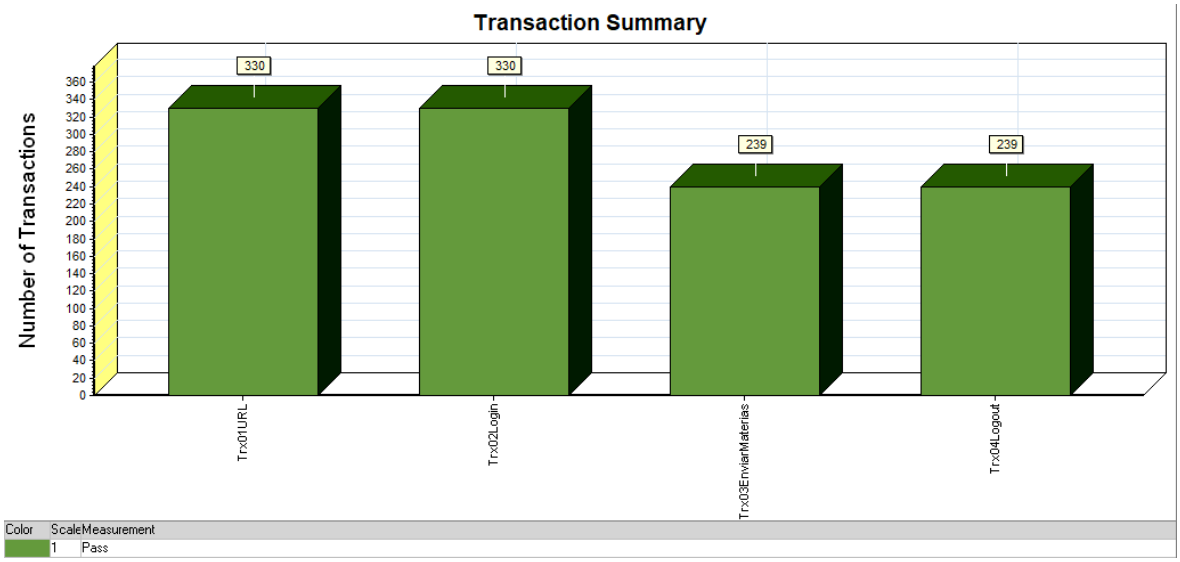


Figura 38. Número de transacciones realizadas por LoadRunner

La **Figura 39** proporciona el tiempo de respuesta promedio para cada una de las transacciones del escenario de prueba.

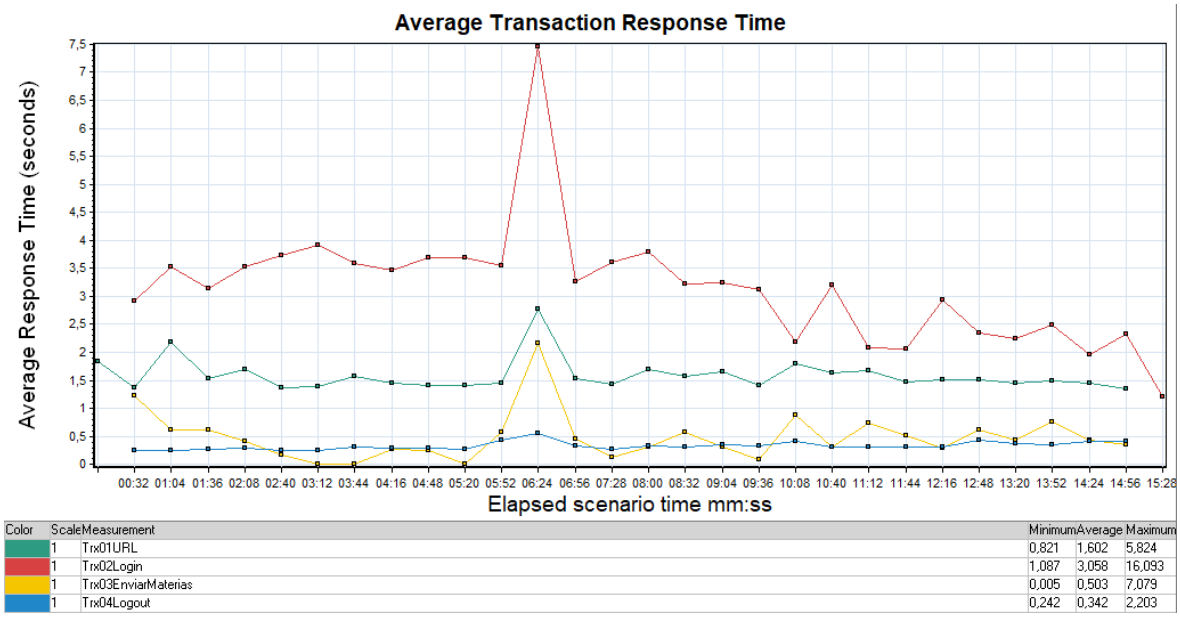
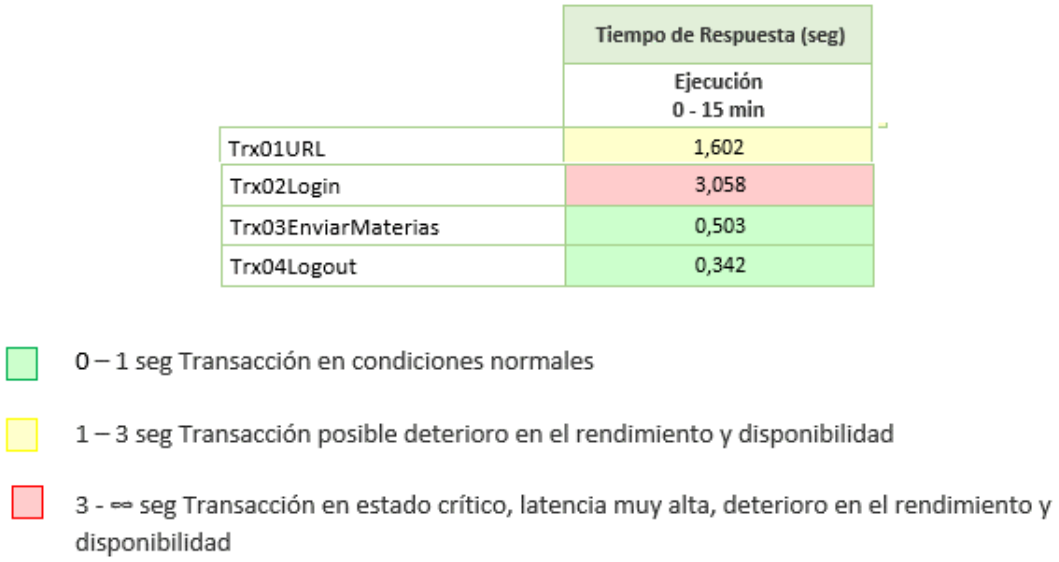


Figura 39. Gráfico de tiempo promedio de respuesta por transacción

En cuanto a la herramienta utilizada para crear la prueba de carga, identificó que el tiempo de respuesta promedio no es óptimo, ya que supera los 3 segundos. Además, la transacción Trx01URL presenta un posible deterioro en el rendimiento y disponibilidad, como se observa en la **Figura 40**.



	Tiempo de Respuesta (seg)
	Ejecución 0 - 15 min
Trx01URL	1,602
Trx02Login	3,058
Trx03EnviarMaterias	0,503
Trx04Logout	0,342

Figura 40. Resultado de la herramienta LoadRunner

Por lo tanto, se confirma el funcionamiento óptimo del sistema, teniendo en cuenta que las capacidades del aplicativo dependen del tipo de suscripción adquirida para el proyecto. Es importante considerar que la prueba con usuarios reales requerirá hacer uso de la elasticidad disponible en cloud computing, ya que la cantidad de usuarios concurrentes en la encuesta variará a lo largo del periodo académico ordinario.

La elección de utilizar Azure DevOps para la automatización del proceso de contrucción del sistema de preplanificación se presenta como una de las opciones más viables debido a las facilidades que ofrece esta herramienta. A lo largo de su implementación, se ha comprobado su practicidad y eficacia en el soporte de las cuatro aristas del sistema.

3.2 Conclusiones

La elección de Azure DevOps para respaldar la entrega continua de incrementos funcionales en el desarrollo del sistema resultó acertada. A pesar de que inicialmente se consideró Jenkins, una opción gratuita, la complejidad de configurar la herramienta aumentó al integrar tecnologías del Back End y Front End, aumentando la curva de aprendizaje y demostrando no ser el camino adecuado para este proyecto. Por lo que se optó por una plataforma diferente, Azure DevOps, la cual redujo la complejidad y permitió la automatización de tareas que ralentizaban la entrega de incrementos funcionales.

La centralización de procesos de integración y despliegue eliminó preocupaciones de compatibilidad y al aprendizaje de nuevas tecnologías. La operación transparente de todas las etapas mediante Azure DevOps facilitó la adición de funcionalidades, priorizando la entrega encima de la mecánica de ejecución. El diseño eficiente del flujo de trabajo desde la concepción hasta la gestión del producto se cumplió exitosamente. Las pruebas, incluyendo la de carga, validaron la solidez de la automatización y construcción de sistemas confiables en un entorno ágil.

El enfoque de Azure DevOps, que permite tanto el funcionamiento individual como en conjunto de los componentes, ha sido fundamental para cumplir con los objetivos de cada sprint, evitando impactos negativos derivados de prácticas de integración y despliegue continuo. La operación transparente en todas las etapas a través Azure DevOps ha facilitado la incorporación de funcionalidades, priorizando la entrega más que la mecánica de ejecución.

El objetivo de diseñar un flujo de trabajo eficiente desde la concepción hasta la implementación y gestión del producto se ha cumplido de manera exitosa. Los resultados obtenidos de las pruebas, incluyendo la prueba de carga, validan la solidez de la automatización y la integración en la construcción de sistemas confiables y eficientes en un entorno de desarrollo ágil y competitivo.

Aunque el proceso de desarrollo del sistema no fue perfecto y presentó inconvenientes, se ha confirmado un funcionamiento óptimo del sistema, el cuál ha sido confirmado por los beneficiarios. Esto se ha logrado al considerar las capacidades del aplicativo y la necesidad de adaptación a la variabilidad de los requisitos del sistema.

La elección de continuar con Azure DevOps para el hospedaje del sistema de preplanificación se respalda en las facilidades que esta herramienta proporciona. A lo largo del proceso, se ha demostrado su practicidad y eficacia en el soporte de todos los aspectos del sistema, consolidando su posición como la opción más viable de todas las que se analizaron.

El diseño y la implementación de un flujo de trabajo interconectado y eficiente ha sido clave para lograr agilidad, calidad y competitividad en el desarrollo de software, respaldados por una herramienta central que ha demostrado su valía en cada etapa del ciclo de vida del producto software.

3.3 Recomendaciones

La elección de utilizar Azure DevOps como eje de integración y entrega continua fue una decisión cuidados, dada su naturaleza de herramienta de pago. Al emplear herramientas con costos de suscripción, es esencial asegurarse de que la suscripción se adapte a las necesidades del proyecto. Es fundamental comprender la infraestructura del proyecto y obtener bien los requerimientos para evitar gastos extras en etapas avanzadas del proyecto.

Es crucial abordar de manera consciente la deuda técnica al iniciar el proceso de desarrollo. Reconocer nuestras limitaciones en una tecnología, como Azure DevOps en este caso, resultó fundamental para tomar decisiones apropiadas para que no impacten en la calidad percibida por el usuario final. Enfrentar la deuda técnica promueve el desarrollo sostenible del software y garantiza que las soluciones a corto plazo no comprometan la mantenibilidad a largo plazo del sistema.

Con sus herramientas como Boards, Azure DevOps facilitó una asignación eficiente de esfuerzos entre los miembros del equipo a medida que avanzaba el desarrollo, asegurando así el éxito del proyecto. Esta plataforma permite focalizar los esfuerzos en tareas que contribuyan directamente a los objetivos, evitando la realización de trabajo innecesario. La habilidad para asignar tiempo y recursos de manera precisa, evitando redundancias, es de vital importancia. Esta estrategia maximiza la utilización de la experiencia de cada miembro para impulsar proactivamente el proyecto hacia adelante.

La fusión de los valores esenciales de Scrum con las prácticas de integración y entrega continua influye significativamente en la adaptabilidad del equipo de desarrollo. Enfocarse a la transparencia, la inspección y la adaptación capacita a los equipos para abordar cambios en los requisitos. Mantener estos valores de manera constante sitúa al equipo en

una posición favorable para manejar transiciones y cambios, lo que en última instancia se traduce en una gestión del cambio más fluida y en resultados de proyecto más exitosos.

Una lección clave es la importancia de la comunicación efectiva para superar obstáculos y alcanzar objetivos de equipo. Fomentar un ambiente donde los miembros puedan compartir abiertamente los desafíos impulsa una mentalidad colaborativa para resolver problemas, destacando el valor de herramientas como Azure DevOps, que centralizan la información. La disposición a buscar asistencia cuando sea necesario promueve la colaboración y mejora continua. Este enfoque previene bloqueos potenciales, facilita la transferencia de conocimientos y acelera el avance hacia los hitos del proyecto.

Finalmente, la estimación precisa de las tareas resulta crucial para la planificación y ejecución del proyecto. La inclinación a subestimar tareas basadas en su aparente simplicidad puede causar demoras imprevistas y obstaculizar el avance. Reconocer la complejidad requerida para implementar funcionalidades evita subestimar la carga de trabajo, establece plazos realistas, evita apresuramientos y asegura una progresión fluida de las tareas.

4 REFERENCIAS BIBLIOGRÁFICAS

Beck, K., Beedle, M., Bennekum, A. Van, & Cockburn, A. (2001). Manifesto for agile software development. *Wiley Online Library*, *41*(9), 943–944. https://doi.org/10.1002/spe.1100

Chen, L. (2015). Continuous Delivery: Huge Benefits, but Challenges Too. *IEEE Software*, *32*(2), 50–54. https://doi.org/10.1109/MS.2015.27

Jabbari, R., bin Ali, N., Petersen, K., & Tanveer, B. (2016). What is DevOps? A Systematic Mapping Study on Definitions and Practices. *Proceedings of the Scientific Workshop Proceedings of XP2016*. https://doi.org/10.1145/2962695.2962707

Letelier, P., & Penadés, M. C. (2017). *El conflicto en el agilismo: una perspectiva desde el SCRUM*. https://repository.eafit.edu.co/handle/10784/11766

Rossberg, J. (2019). An Overview of Azure DevOps. *Agile Project Management with Azure DevOps*, 37–66. https://doi.org/10.1007/978-1-4842-4483-8_2

Sánchez Peño, J. M. (2015). *Pruebas de Software. Fundamentos y Técnicas*. https://oa.upm.es/40012/

Serna M., E., Martínez M., R., & Tamayo O., P. (2019). A Review to Reality of Software Test Automation. *Computación y Sistemas*, *23*, 169–183.

Soh, J., Copeland, M., Puca, A., & Harris, M. (2020). Continuous Integration/Continuous Delivery with Azure DevOps. *Microsoft Azure*, 493–519. https://doi.org/10.1007/978-1-4842-5958-0_21

Zhu, L., Bass, L., & Champlin-Scharff, G. (2016). DevOps and Its Practices. *IEEE Software*, *33*(3), 32–34. https://doi.org/10.1109/MS.2016.81

5 ANEXOS

ANEXO I

Enlace a un vídeo donde se muestra una breve explicación del sistema implementado:

https://youtu.be/vySgI_56F7A