INSTRUCTIVO GESTIÓN DE CÓDIGO FUENTE DE LOS PROGRAMAS Y LAS LIBRERÍAS

1. PROPOSITO

Definir los lineamientos y políticas del Instructivo gestión de códigos fuentes de los programas y las librerías definido en la OIA.

2. ALCANCE

Aplica para toda la plataforma de tecnología en todos sus ambientes tales como; productivos, desarrollo y pruebas.

3. GLOSARIO

- **Ambiente de Pruebas:** Infraestructura de hardware y software necesaria para realizar la ejecución de pruebas de integración, sistema y funcionales para determinar el comportamiento de la aplicación en escenarios reales.
- **Ambiente de desarrollo:** Es un conjunto de procedimientos y herramientas que se utilizan para desarrollar un código fuente o programa. Este ambiente es utilizado para escribir, generar, probar y depurar un programa.
- **Ambiente de Producción:** El ambiente de Producción es donde están disponibles las funcionalidades necesarias tecnológicas y de procesos para la ejecución de los procedimientos.
- **Control de código fuente:** El control de código fuente (o el control de la versión) es la práctica de seguimiento y administración de los cambios en el código durante el proceso de desarrollo.
- **Git:** Es un sistema de control de versiones distribuido, lo que significa que un clon local del proyecto es un repositorio de control de versiones completo. Estos repositorios locales plenamente funcionales permiten trabajar sin conexión o de forma remota con facilidad.
- **Branch:** Una rama o branch es una versión del código del proyecto sobre el que se está trabajando. Estas ramas ayudan a mantener el orden en el control de versiones y manipular el código de forma segura.
- **Rama main (Master):** Por defecto, el proyecto se crea en una rama llamada Main (anteriormente conocida como Master). Cada vez que añades código y guardas los cambios, estás haciendo un commit, que es añadir el nuevo código a una rama. Esto genera nuevas versiones de esta rama o branch, hasta llegar a la versión actual de la rama Main.
- **Gitlab:** Es una plataforma Git y DevOps basada en la nube que ayuda a los desarrolladores a supervisar, probar y desplegar su código. Permite gestionar, administrar, crear y conectar los repositorios con diferentes aplicaciones y hacer todo tipo de integraciones, ofreciendo un ambiente y una plataforma para realizar varias etapas de su SDLC/ADLC y DevOps.

- **DevOps**: Es una metodología de trabajo que, basada en el desarrollo de código mediante el uso de nuevas herramientas y prácticas, reduce la tradicional distancia entre técnicos de programación y de sistemas. Este nuevo enfoque de colaboración permite a los equipos trabajar de forma más cercana, aportando mayor agilidad al negocio y notables incrementos de productividad.
- **Microsoft Azure:** Es una plataforma de pago por uso que integra servicios completos en la nube pública para que desarrolladores y equipos de TI administren e implementen aplicaciones y otros recursos a través de un gran centro de datos mundial.
- **Azure repos:** Es el conjunto de herramientas de control de versiones que se puede usar para administrar el código.
- **Pipelines compilación:** Son la forma en que se compila, prueba y empaqueta el código.
- **Pipelines liberación:** Se tratan de implementar ese código en los entornos
- **Plan de pruebas (test plans):** Proporciona herramientas enriquecidas y eficaces que todos los miembros del equipo pueden usar para impulsar la calidad y la colaboración en todo el proceso de desarrollo.
- **Artefactos (Artifacts):** Permite a los desarrolladores almacenar y administrar sus paquetes y controlar con quién quieren compartirlos. Los artefactos de canalización se generan después de compilar su aplicación
- **Put**: La petición HTTP PUT crea un nuevo elemento o reemplaza una representación del elemento de destino con los datos de la petición. es un método idempotente: llamarlo una o más veces de forma sucesiva tiene el mismo efecto (sin efectos secundarios).
- **Post**: Una sucesión de peticiones POST idénticas pueden tener efectos adicionales, como enviar una orden varias veces.
- **Ramas**: Son referencias ligeras que mantienen un historial de confirmaciones y proporcionan una manera de aislar los cambios de una característica o una corrección de errores de la rama principal y otro trabajo.

4. RESPONSABILIDAD Y AUTORIDAD

Líder del proceso desarrollo de software.

5. POLITICAS DE OPERACIÓN:

De acuerdo con el Manual de políticas de seguridad digital:

El gestor de desarrollo y el desarrollador del proyecto son los únicos que pueden tener acceso al código fuente de las aplicaciones que se realizan en la Oficina Asesora de Informática. En ningún motivo estas cláusulas pueden ser cedidas a terceros, ni funcionarios ni contratistas sin la debida autorización del jefe de la oficina.

Las credenciales de acceso (usuario y contraseña) sobre los aplicativos las otorga mesa de ayuda a través de infraestructura, mediante el formato control de accesos GTIGPS02-F005, éstas son otorgadas al gestor y desarrollador, tienen por objeto garantizar que únicamente sean ellos, de acuerdo con su perfil y funciones, quienes tengan acceso al código fuente de las aplicaciones.

Al momento de solicitar un servidor en pruebas o un servidor nuevo para poner en producción el responsable de dar acceso a los servidores, área de infraestructura.

El proceso de solicitud se debe realizar de la siguiente manera:

- Mesa de servicio recibe la solicitud de los usuarios a través del aplicativo SAUS
- El coordinador de la mesa de servicio asigna la solicitud al área de infraestructura o área de desarrollo según el requerimiento
- El técnico o analista de infraestructura revisa, analiza y autoriza la remediación de una vulnerabilidad existente detectada por el área de seguridad
- Mesa de SAUS informa al solicitante si la solicitud no es aceptada
- Si es aceptada mesa de SAUS informa al solicitante y se cierra la mesa SAUS
- Mesa de SAUS redirecciona a mesa de ayuda al área de soporte.
- El técnico encargado realiza la instalación de acuerdo con el manual de usuario e ingresa el seguimiento en la Mesa de servicios
- Soporte tecnico verifica el correcto funcionamiento del software y actualiza el seguimiento de Mesa de SAUS
- Soporte tecnico verifica el correcto funcionamiento del software y actualiza el seguimiento de Mesa de SAUS
- El técnico correspondiente se pone en contacto con el usuario para realizar Capacitación del software
- Cierre del caso en el aplicativo SAUS

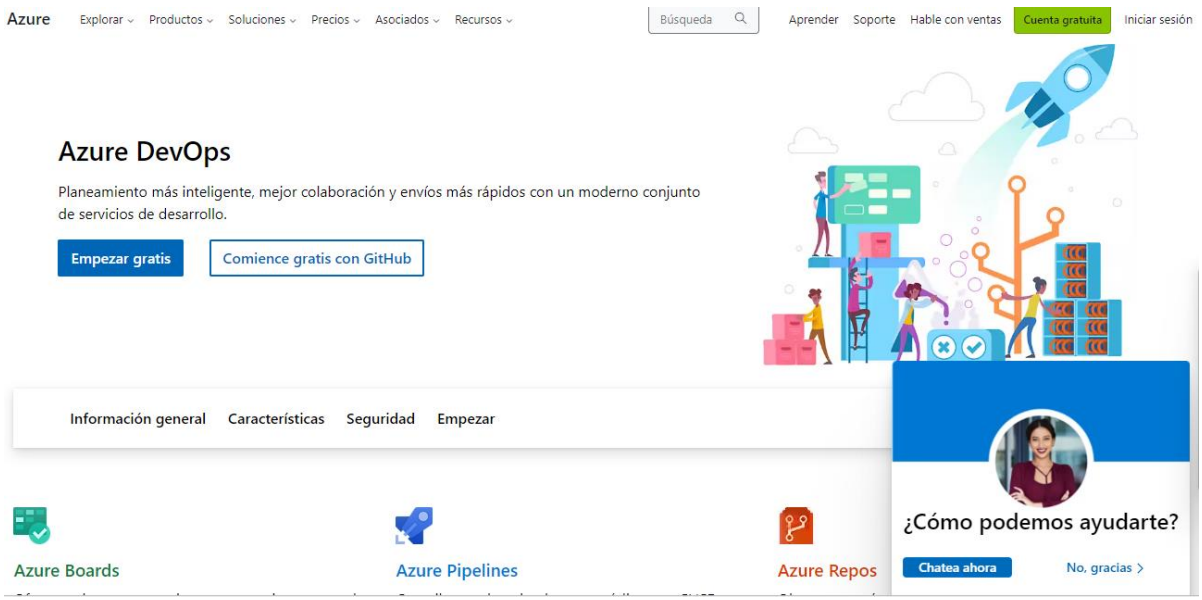
Todos los cambios, modificaciones, mejoras o actualizaciones que se efectúen al código, deberán estar debidamente documentados y registrados en los formatos ya definidos por la Oficina Asesora de Informática.

Los pasos a producción de los cambios, modificaciones, mejoras o actualizaciones que se efectúen al código, se realizarán con la autorización del gestor de desarrollo.

CREACIÓN DE UN PROYECTO EN AZURE DEVOPS

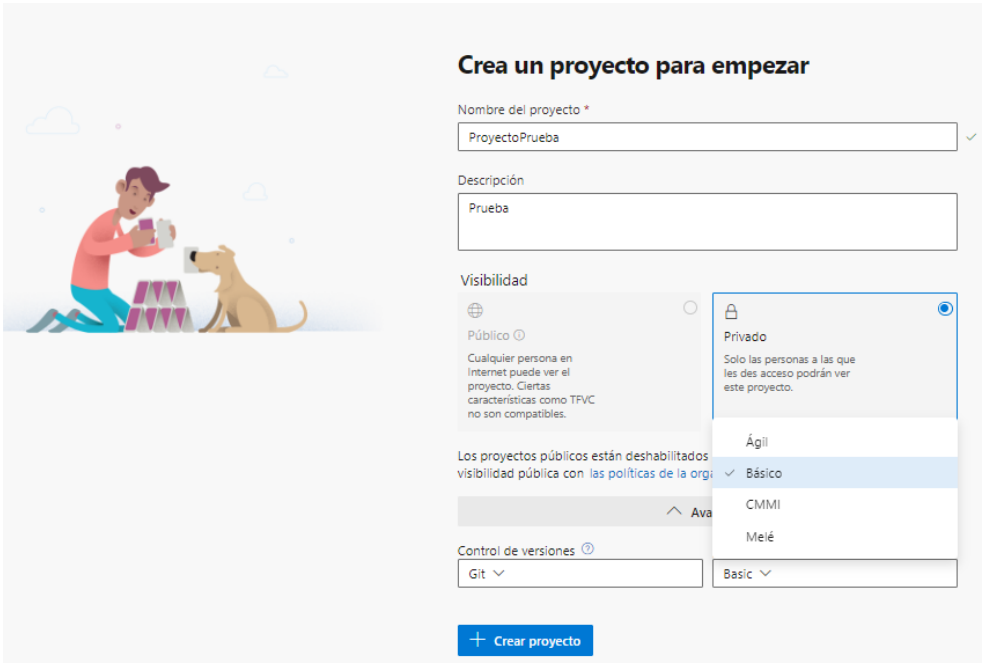
Pasos.

1. Entrar en la página de AzureDevOps https://azure.microsoft.com/es-mx/products/devops/.
2. Presionar clic en Inicio Gratuito
3. Revisar los términos del servicio, el aviso de privacidad y el código de conducta.
4. Presionar clic en Continuar (Empezar gratis)



Gráfica 1: Aceptación de términos del servicio

5. Se creará una organización con el nombre del correo, y preguntará el nombre del proyecto y la opción de ponerlo público o privado. Como ejemplo al nombre se le colocó, ProyectoPrueba y se dejará público.



Gráfica 2: Creación del proyecto

6. En la sección de Advanced se podrá configurar que tipo de Control de Versiones que se va a manejar, y que tipo de metodología se va a utilizar.

- **Agile:** Si se va a utilizar metodologías ágiles y si se quiere registrar actividades de desarrollo, testing y bugs.

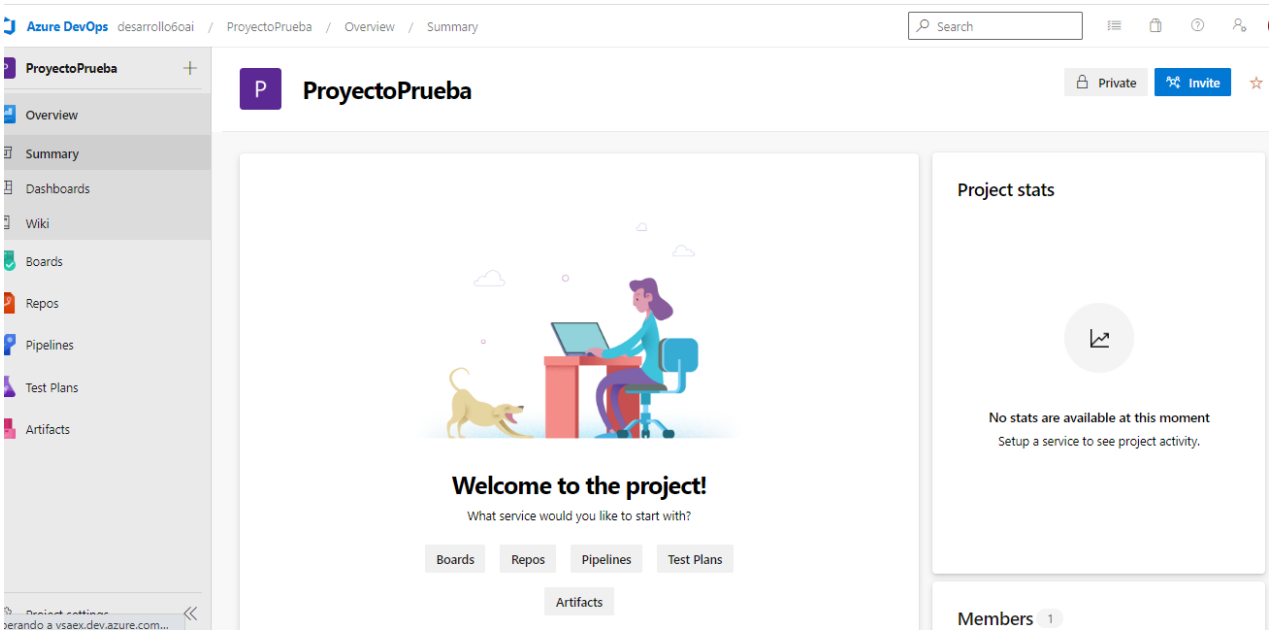
- **Scrum**: Permite registrar las tareas y bugs en un Kanboard, el cual es una forma de representar en forma de columnas las tareas pendientes y las terminadas.

En este caso se usará la metodología Scrum y GIT

7. Presionar clic en Create Project

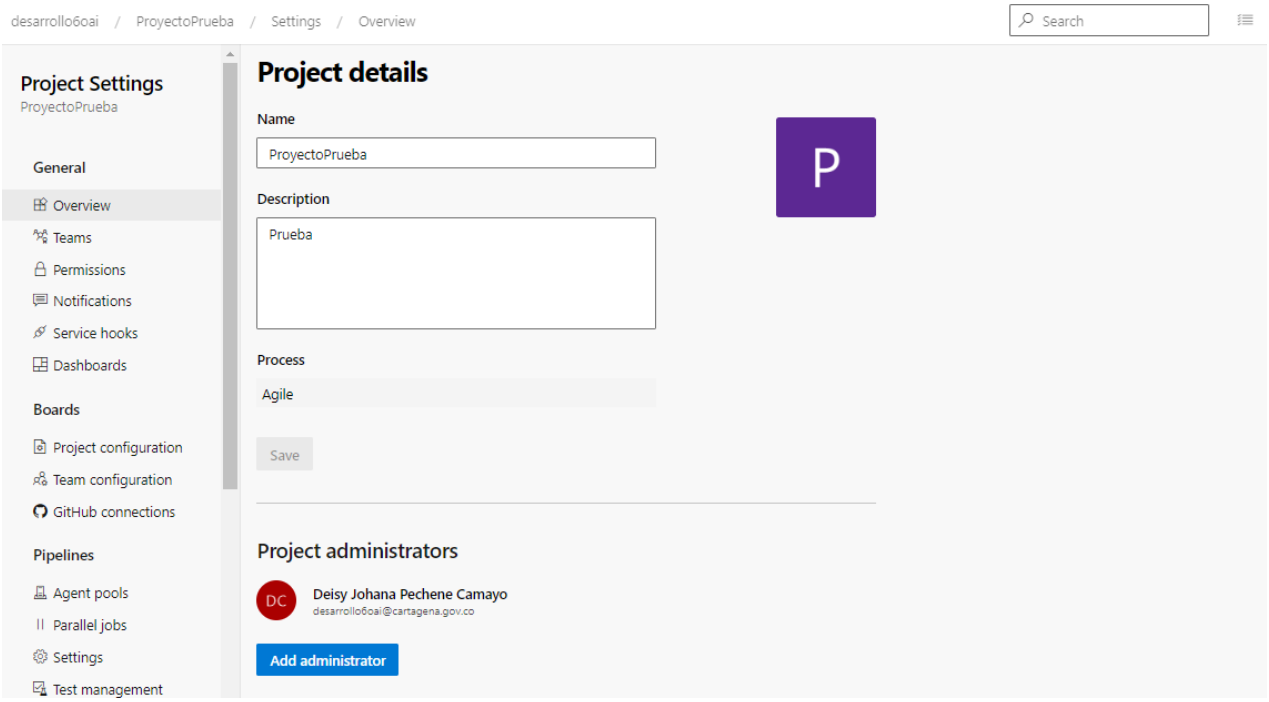
Esperar unos segundos en lo que se crea el proyecto.

8. Del lado izquierdo se encuentran las opciones del proyecto de forma general, en la parte central muestra una sugerencia de las actividades que se pueden realizar y del lado izquierdo se muestran las estadísticas del proyecto.



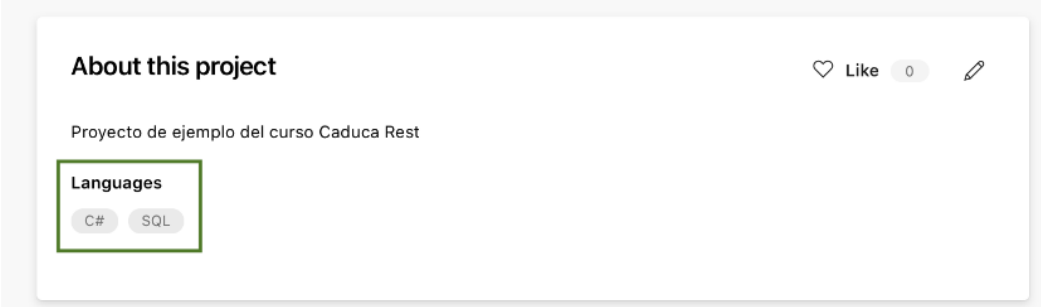
Gráfica 3: Actividades y estadísticas

9. Dar clic en Overview para agregar la descripción del proyectoDa clic en Overview para agregar la descripción del proyecto.



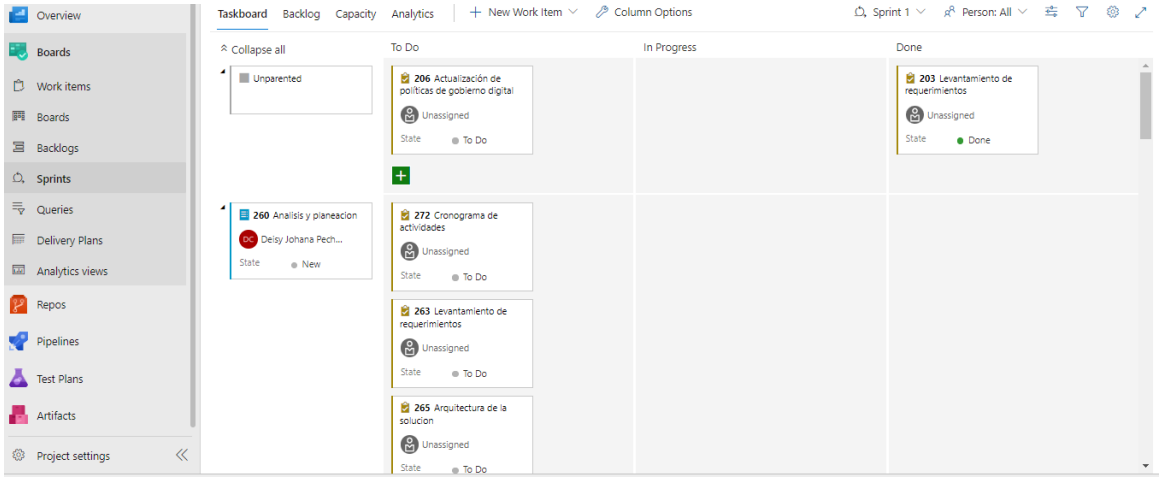
Gráfica 4: Descripción del proyecto

10. Después cuando se agrega el código se muestran los lenguajes que se está utilizando en el proyecto, en este caso es C# y SQL. Después cuando se agrega el código se muestran los lenguajes que utilizará en el proyecto, en este caso es C# y SQL.



Gráfica 5: Selección de lenguaje de programación

11. Se procede a crear los sprint, las actividades correspondientes



Gráfica 6: Creación de sprint

CREACIÓN DE LA RAMA DESARROLLO EN AZURE DEVOPS

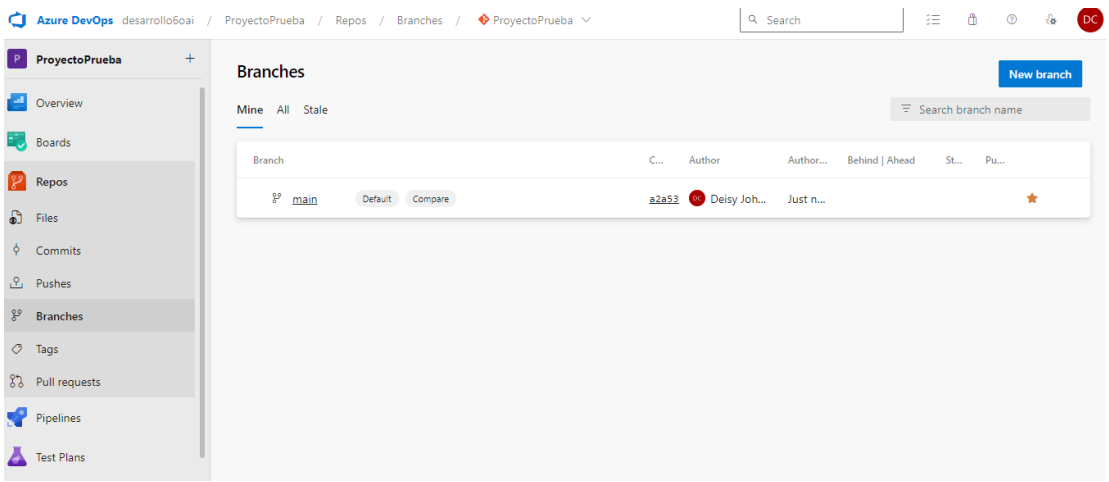
Pasos.

Se puede crear una rama de desarrollo en base a la rama master para que todos los programadores hagan sus cambios sobre la rama desarrollo.

1. En el explorador web, abrir el proyecto de equipo de la organización de Azure DevOps y, a continuación, elejir:

2. Repos Branches (Ramas de repositorios>) para abrir la vista Ramas.

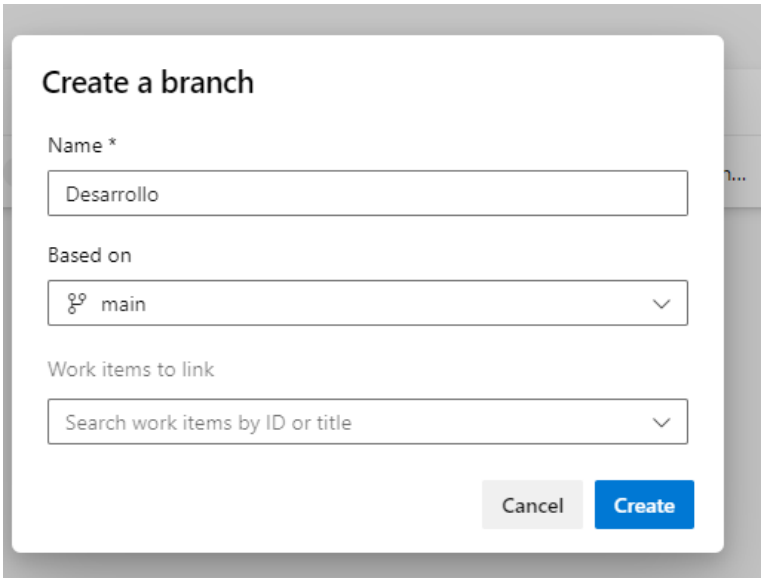
3. En la vista Ramas, elegir Nueva rama para iniciar el cuadro de diálogo Crear una rama.



Gráfica 7: Creación de ramas

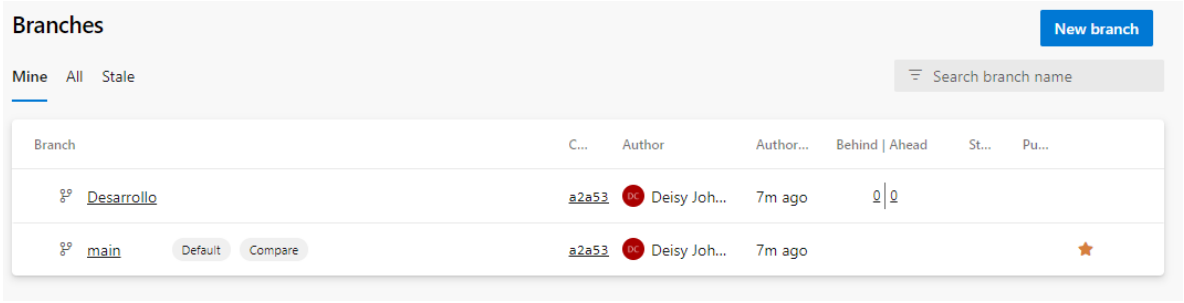
4. Configurar lo siguiente:

   - Name: El nombre de la nueva rama.
   - Teclear desarrollo.
   - Based on: Indica la rama de la cual se va a copiar los archivos. En este caso es master.
   - Works items to link: Se puede agregar el user story, o la tarea para esta nueva rama. Como es la rama inicial de desarrollo no se va a selecciona nada.
   - Dar clic en Create de esta manera se crea la rama desarrollo con una copia del código de la rama master, que es la rama del código que está en producción.



Gráfica 8: Configuración de ramas

5. La nueva rama aparecerá en la lista de ramas

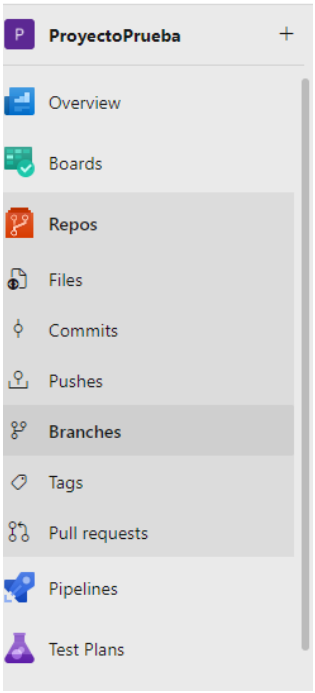


Gráfica 9: Lista de ramas

ELIMINAR RAMA DE GIT

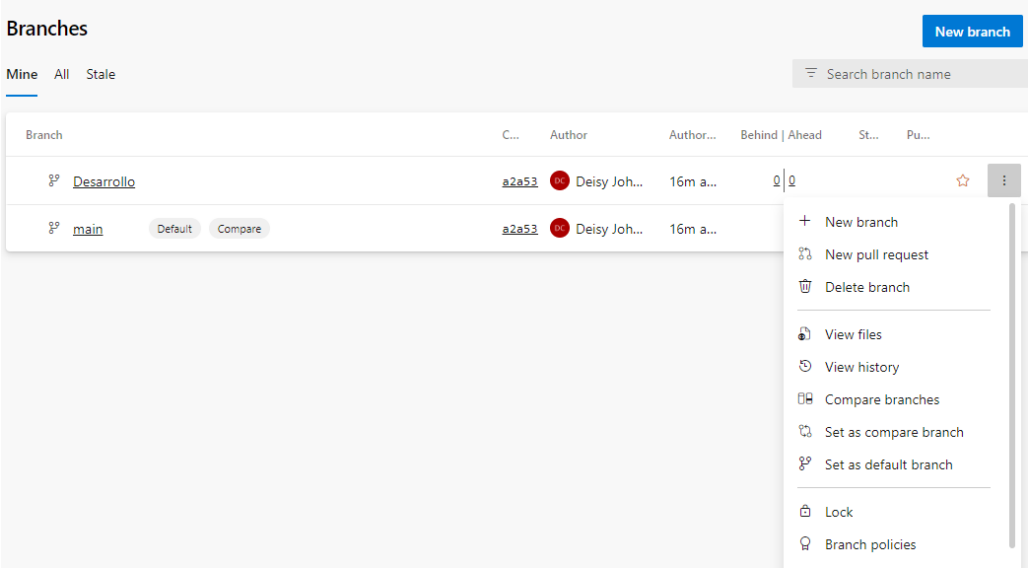
Pasos.

1. Para ver las ramas del repositorio, seleccionar Ramas de repositorio> mientras se visualiza el repositorio en la Web.



Gráfica 10: Rama de repositorio

2. Seleccionar el botón Más opciones, al final de la fila de la rama que se desea eliminar.



Gráfica 11: Configuración para eliminar rama

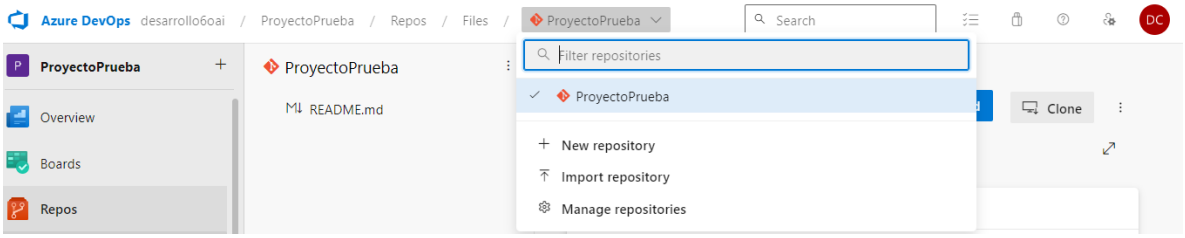
3. En el menú de opciones, seleccionar Eliminar rama.

4. En el cuadro de diálogo Eliminar rama, seleccionar Eliminar.

CREACIÓN DE UN REPOSITORIO DE GIT EN EL PROYECTO
Pasos.

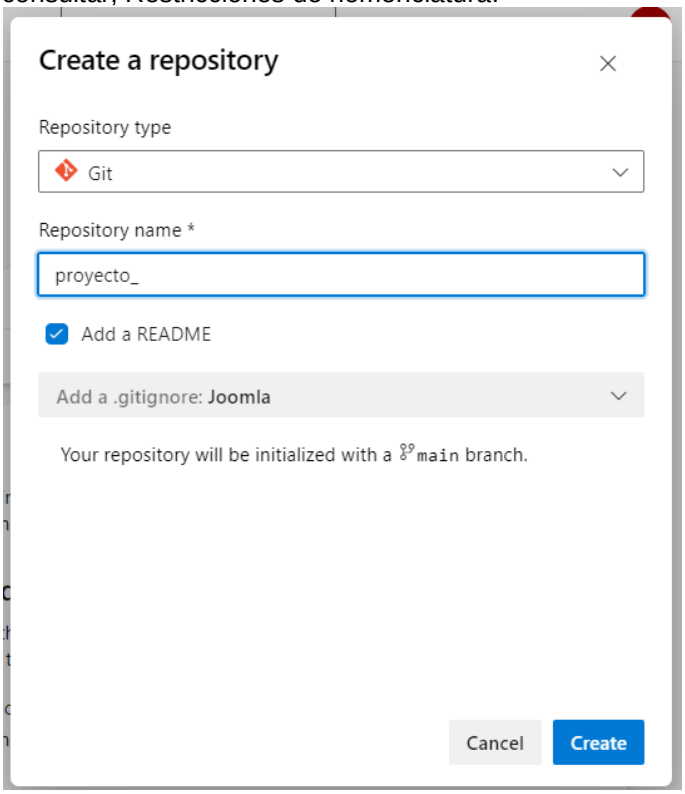
1. Abrir la página Repos del proyecto

2. Seleccionar el nombre del proyecto, en la lista desplegable repositorio, seleccionar icono New Repository



Gráfica 12: Selección de icono New Repository

3. En el cuadro de diálogo Crear un repositorio, comprobar que Git es el tipo de repositorio y escribir un nombre para el nuevo repositorio. Para conocer las restricciones de nomenclatura, consultar, Restricciones de nomenclatura.



Gráfica 13: Configuración repositorio

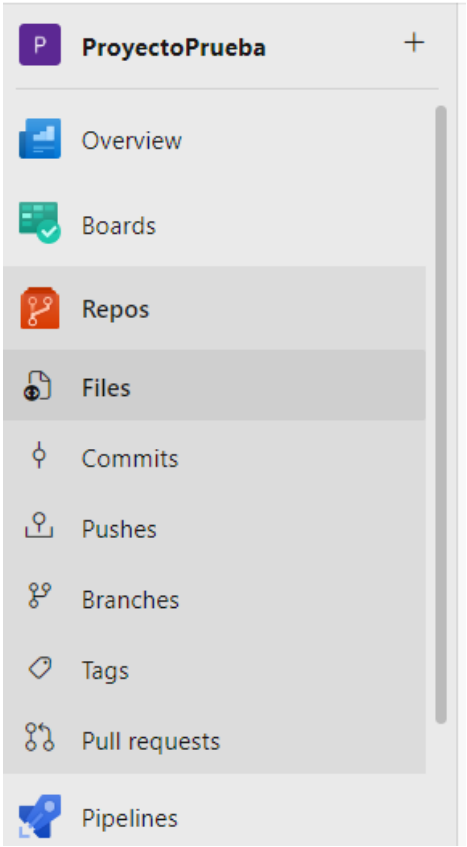
4. También se puede agregar un ARCHIVO LÉAME y crear un archivo. gitignore para el tipo de código que se planea administrar en el repositorio.
5. Un ARCHIVO LÉAME contiene información sobre el código del repositorio. El archivo. gitignore indica a Git qué tipos de archivos omitir, como los archivos de compilación temporales del entorno de desarrollo.
6. Cuando los datos están correctos en el nombre y las opciones del repositorio, seleccionar Crear.
7. Ahora se crea un nuevo repositorio de Git vacío en el proyecto.

CLONACIÓN DEL REPOSITORIO EN EL EQUIPO

Pasos.

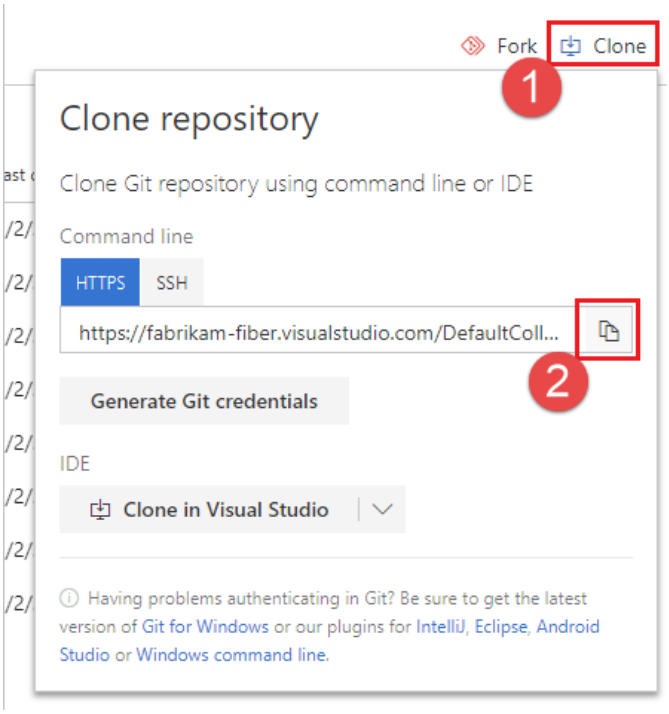
1. Para trabajar con un repositorio de Git, se clona en el equipo. La clonación de un repositorio crea una copia local completa del repositorio con la que trabajar. La clonación también descarga todas las confirmaciones y ramas del repositorio y configura una relación con nombre con el repositorio en el servidor. Usar esta relación para interactuar con el repositorio existente, insertar y extraer cambios para compartir código con el equipo.

2. En el explorador web, abrir el proyecto de equipo de la organización en Azure DevOps y seleccionar Archivos de repositorios>. Si no se tiene un proyecto de equipo se deberá crear uno.



Gráfica 14: Configuración archivo repositorios

3. Seleccionar Clonar en la esquina superior derecha de la ventana Archivos y copiar la dirección URL de clonación.



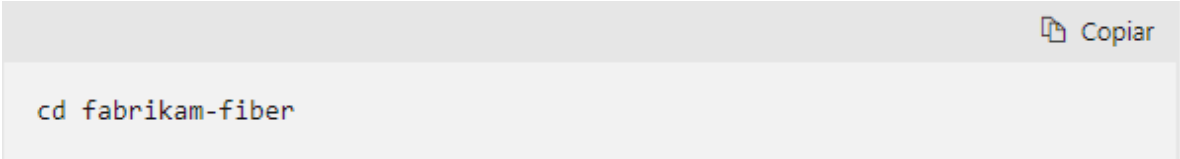
Gráfica 15: Configuración archivo a clonar

4. Abrir la ventana de comandos de Git (Git Bash en Git para Windows). A continuación, ir a la carpeta donde desea que el código del repositorio se almacene en el equipo. Ejecutar git clone seguido de la ruta de acceso copiada de la dirección URL de clonación de la sección anterior.

Gráfica 16: Ejecución comando GIT

Git descarga una copia del código en una nueva carpeta con la que trabaja. La descarga incluye todas las confirmaciones y ramas del repositorio.

5. Cambiar el directorio al repositorio que ha clonado.



Gráfica 17: Cambio de directorio

CONFIGURACIÓN DEL GITHUB

Pasos.

1. En el explorador web, ir a la página principal del repositorio de GitHub
2. Seleccionar una rama base para iniciar el cuadro de diálogo Cambiar ramas o etiquetas
3. Escribir un nombre de rama nuevo único y, a continuación
4. Elegir: Crear rama.

CREACIÓN DEL PULL REQUEST

Para crear el Pull Request desde Azure DevOps se puede realizar de las siguientes maneras:

Opción 1:
Pasos:

1. Entrar a la cuenta de Azure DevOps, luego dar clic en Pull requests.
2. En la parte superior se muestra el Pull Request que se creó en la sección anterior.

Opción 2:
Pasos:

1. En la cuenta de Azure DevOps dar clic en Commits, en la rama seleccionada la cual se ve en la parte superior izquierda, al lado del menú principal.
2. En la sección superior se muestra tu Pull Request.
3. Dar clic en Create a pull rquest.
4. Configurar lo siguiente:

- Rama: En la parte superior elegir en que rama deseas agregar tus cambios.

Por lo general se agregan a la rama desarrollo.

En este ejemplo se agregará a la rama desarrollo.

5. Title: Elegir un título descriptivo del cambio, o puede seguir alguna convención de nombres.

6. Description: Puede agregar una descripción más detallada de los cambios.

7. Reviewers: Este elije qué programador o persona va a revisar los cambios de código.

8. Work items to link: Se puede enlazar el número de task relacionado con el código

9. Files: Se muestra la lista de archivos con cambios y se selecciona para ver los cambios.

10. Opción Files: Del lado izquierdo se muestra la lista de archivos izquierdo. En el ejemplo solo se cambiará el archivo Categoria.cs y se cambiará unos comentarios.

11. Dar clic al archivo y se muestra del lado izquierdo como estaba el código antes y como estaba después.

12. Puede volver a revisar para comparar que no se haya olvidado algo, o algún error de escribir mal una variable. etc.

Opción 3.
Pasos:

1. Dar clic en Create.

2. Según como estén configuradas las notificaciones en el proyecto, se le enviará un correo al programador que se eligió como reviewer, para estar enterado de que debe revisar el cambio.

3. Se puede agregar un programador por default que revise cada pull request, o se puede por ejemplo poner reglas para que no se elija la rama master por error.

4. Se muestra las siguientes opciones:

    - Files: Se muestran los archivos cambiados

    - Updates: En caso de que el programador que revisa tenga sugerencias, los nuevos cambios se muestran en la sección Updates.

    - Commits: Se muestra la descripción que se le dio a los commits.

    - Approve: Si no se tiene reviewer obligatorio, se podrá Aprobar el cambio en esta opción. De lo contrario solo le aparecerá al Reviewer cuando se consulta tu Pull Request.

    - Complete: Permite completar el cambio e integrarlo a la rama elegida. En este ejemplo es desarrollo.

5. En la parte inferior se ven los comentarios que hacen los reviewers, puede hacerlos en esa sección o directamente en el código.

6. Una vez que los cambios son aprobados dar clic en Complete para incorporar los cambios a la rama desarrollo. Una vez que los cambios son aprobados dar clic en Complete para incorporar los cambios a la rama desarrollo.

6. SEGURIDAD DE CONTROL DE VERSIONES

Para garantizar la seguridad en el control de versión, se deben considerar los siguientes aspectos:

Autenticación: Es necesario asegurar que solo las personas autorizadas puedan acceder al sistema de control de versión. Esto implica la implementación de autenticación sólida, como contraseñas seguras o autenticación de dos factores.

Autorización: Se deben establecer permisos y roles adecuados para controlar el acceso a los diferentes repositorios y recursos del sistema de control de versión. Esto ayuda a prevenir modificaciones no autorizadas o manipulaciones indebidas.

Encriptación: Los datos almacenados en el sistema de control de versión deben ser encriptados, tanto en tránsito como en reposo, para proteger su confidencialidad y evitar la exposición a posibles ataques.

Auditoría y registro de actividad: Se deben mantener registros detallados de las actividades realizadas en el sistema de control de versión, incluyendo quién realizó cambios, cuándo se realizaron y qué cambios se hicieron. Esto ayuda a detectar y rastrear cualquier actividad sospechosa o no autorizada.

Respaldo y recuperación: Es importante realizar copias de seguridad periódicas de los repositorios y tener planes de recuperación en caso de pérdida de datos o fallos del sistema. Esto garantiza la disponibilidad de los activos y la capacidad de restaurar versiones anteriores si es necesario.

Actualizaciones y parches: Mantener el sistema de control de versión actualizado con las últimas actualizaciones de seguridad y parches es fundamental para prevenir vulnerabilidades conocidas y asegurar un entorno seguro.

Educación y concientización: Los usuarios del sistema de control de versión deben recibir capacitación adecuada sobre las mejores prácticas de seguridad, como el uso de contraseñas fuertes, la protección de credenciales y la prevención de ataques de ingeniería social.

Al momento de finalizar el proceso de actualizaciones el código fuente se almacena y su destino es en el repositorio de Azure DevOps, y en el servidor donde se ejecutará la aplicación.

Al finalizar el desarrollo del aplicativo y al finalizar el proceso de pruebas el aplicativo se mueve a un servidor en producción, en donde queda el código y el repositorio, en el repositorio queda un código limpio, sin errores, que no va a tener registros. En el servidor si se tiene registros, ya que se va a tener una conexión a bases de datos.

Responsable y custodio del repositorio área de desarrollo.
Responsable y custodio del acceso a servidores área de infraestructura.

7. PROCEDIMIENTO DE GESTIÓN DE CAMBIOS:

La gestión de cambios en el desarrollo de sistemas de información es un proceso crítico para garantizar que los cambios en un sistema se realicen de manera controlada y efectiva sin causar interrupciones o problemas no deseados. A continuación, se presenta un procedimiento general para la gestión de cambios en el desarrollo de sistemas de información:

7.1 Identificación del cambio:

El primer paso es identificar y documentar el cambio propuesto. Esto puede ser una nueva característica, una corrección de error, una mejora, etc. Es importante especificar claramente qué se va a cambiar y por qué se necesita el cambio.

7.2 Evaluación del impacto:

Se debe evaluar el impacto potencial del cambio en el sistema existente. Esto incluye analizar cómo afectará a la funcionalidad existente, a los datos y a otros componentes del sistema. Se debe determinar si el cambio es factible y si es necesario realizar pruebas adicionales.

7.3 Creación de una solicitud de cambio:

Una vez que se haya evaluado el impacto, se debe crear una solicitud formal de cambio. Esta solicitud debe incluir detalles como la descripción del cambio, la justificación, los riesgos asociados y cualquier otro dato relevante.

7.4 Revisión y aprobación:

La solicitud de cambio debe ser revisada por un comité de revisión de cambios o un grupo similar. Este comité evaluará la solicitud y tomará una decisión sobre si aprobar o rechazar el cambio. La aprobación debe basarse en criterios como la importancia del cambio, el impacto en el sistema y la disponibilidad de recursos.

7.5 Planificación del cambio:

Una vez que se haya aprobado el cambio, se debe desarrollar un plan detallado para implementarlo. Esto incluye la asignación de recursos, la definición de un cronograma y la identificación de las etapas necesarias para llevar a cabo el cambio.

7.6 Desarrollo y pruebas:

Llevar a cabo el desarrollo del cambio de acuerdo con el plan establecido. Realizar pruebas exhaustivas para asegurarse de que el cambio funcione correctamente y no cause problemas en otras partes del sistema.

7.7 Implementación:

Una vez que se haya probado el cambio de manera satisfactoria, se puede implementar en el entorno de producción. Esto puede requerir una interrupción programada o un proceso de implementación gradual, según la naturaleza del cambio.

7.8 Seguimiento y control:

Después de la implementación, es importante realizar un seguimiento del cambio para asegurarse de que funcione como se esperaba y que no cause problemas inesperados. También es fundamental tener un plan de contingencia en caso de que surjan problemas después de la implementación.

7.9 Documentación:

Se debe mantener una documentación detallada de todos los cambios realizados, incluyendo la justificación, los resultados de las pruebas y cualquier otro detalle relevante.

7.10 Cierre:

Una vez que el cambio se haya implementado y se haya comprobado que funciona de manera efectiva, se puede cerrar el proceso de gestión de cambios.

8. MECANISMOS PARA LA ENTREGA CONTINUA DE SISTEMAS DE INFORMACIÓN

La entrega continua (CD, por sus siglas en inglés, Continuous Delivery) es una práctica que se basa en automatizar el proceso de entrega de software de manera que se pueda liberar nuevas versiones del sistema de información de manera eficiente y confiable. A continuación, se menciona algunos de los mecanismos y prácticas claves utilizados para lograr la entrega continua en sistemas de información:

8.1 Automatización de la Construcción y Empaquetado: Utiliza herramientas como Apache Maven, Gradle o sistemas de construcción específicos para tu tecnología para automatizar la compilación y empaquetado de la aplicación. Esto asegura que el código se compile de manera consistente y que los artefactos estén listos para ser desplegados.

8.2 Control de Versiones: Se debe asegurar que se mantenga un sistema de control de versiones (Ejemplo, Git) para gestionar y rastrear cambios en el código fuente. Las ramas de desarrollo y liberación deben estar claramente definidas.

8.3 Pipeline de Entrega Continua: Se debe crear un pipeline de entrega continua que automatice el flujo de trabajo desde la integración hasta la entrega. Esto incluye la construcción, las pruebas, el análisis estático de código, la revisión de calidad y la implementación.

8.4 Automatización de Pruebas: Implementar pruebas automatizadas a lo largo del pipeline de entrega continua. Esto incluye pruebas unitarias, pruebas de integración, pruebas de aceptación del usuario y pruebas de regresión. Las pruebas automatizadas garantizan la calidad del software y permiten detectar problemas de manera temprana.

8.5 Gestión de Configuración: Utilizar herramientas de gestión de configuración para garantizar que las configuraciones de los entornos de desarrollo, pruebas y producción sean coherentes y estén bien documentadas.

8.6 Entornos de Pruebas Automatizadas: Crear entornos de pruebas automatizadas que se asemejen lo más posible al entorno de producción. Esto permite realizar pruebas exhaustivas antes de la implementación en producción.

8.7 Despliegue Automatizado: Implementar mecanismos de despliegue automatizado que permitan liberar nuevas versiones del sistema con un solo clic o de manera programada. Herramientas como Docker y Kubernetes son útiles para orquestar despliegues en contenedores.

8.8 Monitoreo y Registro: Implementar herramientas de monitoreo y registro para rastrear el rendimiento de la aplicación en producción. Esto ayuda a detectar problemas de manera proactiva y a realizar ajustes en tiempo real.

8.9 Gestión de Dependencias: Utilizar sistemas de gestión de dependencias para administrar bibliotecas y paquetes utilizados en el proyecto. Se debe asegurar de que las dependencias se mantengan actualizadas y sean compatibles.

8.10 Evaluación y Retroalimentación Continua: Recopilar métricas y datos de usuario para evaluar la eficacia del proceso de entrega continua. Esto permite realizar mejoras continuas en el proceso.

8.11 Control de Versiones de Infraestructura (IaC): Utilizar herramientas de Infrastructure as Code (IaC) como Terraform o Ansible para definir y gestionar la infraestructura de manera automatizada, lo que garantiza que los entornos sean coherentes y replicables.

8.12 Seguridad Automatizada: Implementar herramientas de seguridad automatizada para escanear el código en busca de vulnerabilidades y asegurar que las configuraciones sean seguras.

9. MECANISMOS UTILIZADOS PARA EL DESPLIEGUE CONTINUO: Se enfoca en automatizar y acelerar la entrega de aplicaciones y sistemas de información de manera eficiente y confiable. Para lograr el despliegue continuo de sistemas de información, se utilizan varios mecanismos y herramientas. A continuación, se describe a nivel general de algunos de los mecanismos clave utilizados:

9.1 Control **de Versiones:** El control de versiones es fundamental para llevar un registro de los cambios en el código fuente y la infraestructura. Herramientas como Git permiten a los equipos colaborar y mantener un historial completo de las modificaciones.

9.2 Integración **Continua (CI):** La integración continua implica la automatización de pruebas y compilaciones cada vez que se realiza un cambio en el código fuente. Las herramientas de CI, como Jenkins, Travis CI o CircleCI, permiten ejecutar estas acciones automáticamente, lo que ayuda a identificar y solucionar problemas tempranamente.

9.3 Entrega **Continua (CD):** La entrega continua implica la automatización de la preparación y el empaquetado de la aplicación para su despliegue en entornos de prueba o de producción. Las herramientas de CD, como Docker y Kubernetes, ayudan a crear entornos reproducibles y facilitan la distribución de aplicaciones.

9.4 Automatización **de Pruebas:** Las pruebas automatizadas, como las pruebas unitarias, las pruebas de integración y las pruebas de aceptación, son esenciales para garantizar la calidad del software. Las herramientas de automatización de pruebas, como Selenium, JUnit o Jest, permiten ejecutar pruebas de manera automatizada.

9.5 Orquestación **y Gestión de Contenedores:** Las aplicaciones se empaquetan comúnmente en contenedores para facilitar su despliegue. Herramientas como Docker permiten crear, administrar y orquestar contenedores de manera eficiente.

9.6 Orquestación de Despliegue: Las herramientas de orquestación, como Kubernetes, son cruciales para administrar y escalar aplicaciones en contenedores en entornos de producción. Facilitan el despliegue, la actualización y el escalado automático de aplicaciones.

9.7 Automatización de Infraestructura: La infraestructura como código (IaC) permite definir la infraestructura de manera programática. Herramientas como Terraform y AWS CloudFormation permiten gestionar recursos de infraestructura de forma automatizada.

9.8 Monitoreo y Registro: Para garantizar un despliegue continuo exitoso, es esencial monitorear la salud de las aplicaciones y los sistemas en tiempo real. Herramientas como Prometheus, Grafana y ELK Stack (Elasticsearch, Logstash, Kibana) ayudan a recopilar y visualizar datos de monitoreo y registro.

9.9 Implementación Gradual (Canary Deployment): Esta técnica permite desplegar nuevas versiones de aplicaciones en una pequeña porción de usuarios o servidores para evaluar su rendimiento y estabilidad antes de implementarla completamente.

9.10 Gestión de Configuración: Las herramientas de gestión de configuración, como Ansible o Puppet, ayudan a mantener la configuración de los sistemas de manera coherente y automatizada.

9.11 Pipeline de Entrega Continua: Un pipeline de CI/CD es un conjunto de pasos automatizados que abarca desde la integración continua hasta la entrega continua. Define cómo se construyen, prueban y despliegan las aplicaciones.

9.12 Seguridad de Despliegue Continuo: La seguridad es un aspecto crítico del despliegue continuo. Se deben implementar prácticas de seguridad, como análisis estático y dinámico de código, escaneo de vulnerabilidades y acceso controlado.

10. CICLOS DE PRUEBAS PARA DESARROLLOS PROPIOS:

Los ciclos de prueba son una parte fundamental en el proceso de desarrollo de software, ya sea que se esté trabajando en proyectos propios o en equipo. Estos ciclos aseguran que el software funcione correctamente, cumpla con los requisitos y sea confiable antes de su lanzamiento. A continuación, se menciona un esquema general de cómo se puede llevar a cabo ciclos de prueba en los desarrollos propios:

10.1 Requisitos y planificación de pruebas:

Inicialmente se debe identificar los requisitos del sistema, se debe identificar los requisitos del software, se debe detallar ¿Qué debe hacer el sistema? ¿Cuáles son las características clave del sistema?, posteriormente se debe diseñar un plan de pruebas que incluya los casos de prueba que verificarán que el software cumple con estos requisitos.

Proceso de entrega:

Al momento de iniciar el proceso de desarrollo ya sea nuevo o actualización se debe crear por parte del gestor de desarrollo el repositorio donde se va a almacenar el código fuente (Azure DevOps), éste queda bajo el área de desarrollo, dicho repositorio se actualiza manualmente a medida que el desarrollador va realizando los cambios y ajustes, es decir, cuando el cliente solicita nuevos requerimientos, estos requerimientos deberán quedar plasmados en el formato del levantamiento del requerimiento detallado GTIGS01-F002.

Al finalizar el proyecto el desarrollador hará entrega del código fuente al líder del proceso de desarrollo de aplicaciones, mediante vía correo electrónico se notifica la finalización del desarrollo y la entrega de la documentación respectiva. Al finalizar el desarrollo el código fuente se deberá entregar en el repositorio

Una vez se hace la entrega del desarrollo del aplicativo y el cumplimiento de los requerimientos definidos, se deberá iniciar con el proceso de pruebas, en donde se deberá hacer la verificación y comprobar que el producto o aplicativo cumpla con las expectativas del cliente, se deberá hacer la validación de los criterios de accesibilidad y usabilidad web, definidos en los formatos de, Cumplimiento criterios de accesibilidad GTIGS01-F008, Cumplimiento criterios de usabilidad GTIGS01-F005, pruebas unitarias definido en el documento, Formato Pruebas Unitarias GTIS01- F009, pruebas de integración, pruebas de sistema, pruebas de aceptación, pruebas de rendimiento y carga, pruebas de seguridad. *(Formatos en construcción).*

Las pruebas deberán ser realizadas por un equipo independiente al que realizó el software. El ingeniero de software que desarrolló el sistema no es el más adecuado para llevar a cabo las pruebas de dicho software o aplicativo, ya que inconscientemente tratará de demostrar que el software funciona, y no que no lo hace, por lo que la prueba puede tener menos éxito.

El resultado de la ejecución de las pruebas que se originaron satisfactorias y de los fallos que se detectaron, deberán ser documentados en los formatos ya definidos y establecidos por la Oficina Asesora de Informática.

Una vez que se hayan realizado las pruebas se notificará al gestor del desarrollo el hallazgo de estas, posteriormente el gestor notificará al desarrollador y en conjunto se deberá definir el paso a seguir para el cumplimiento de los criterios de accesibilidad y usabilidad.

El desarrollador deberá hacer una planificación para determinar el tiempo en que se demorará corrigiendo las fallas o vulnerabilidades encontradas, el tiempo o estimación deberán ser plasmados, documentados y actualizados en el formato de cronograma de actividades GTIS01- F010. Posteriormente el desarrollador deberá hacer nueva interacción en el desarrollo del aplicativo, realizando un nuevo desarrollo teniendo en cuenta y atendiendo solamente lo que se identificó en las pruebas.

Al finalizar los cambios y correcciones el desarrollador notificará nuevamente al gestor de desarrollo que se realizaron las correcciones o deberá justificar por qué no es posible realizar las correcciones, se procederá a validar nuevamente si estos ajustes fueron realizados y aplicados correctamente, si se valida y se confirma que todo está correcto el aplicativo pasará a producción.

Una vez el aplicativo pasa a producción se realizarán las pruebas de aceptación directamente con el cliente, en el cual se hace la entrega del producto, se expondrá y se socializará por parte del desarrollador la funcionalidad del aplicativo, el desarrollo de los solicitado y el resultado final.

Si el cliente está de acuerdo y notifica que el aplicativo cumple con sus expectativas el aplicativo pasará a producción, de lo contrario se procederá a hacer una nueva interacción en el desarrollo para poder cumplir con lo solicitado y esperado por el cliente.

La socialización y entrega del aplicativo deberá hacerse de manera presencial o virtual sea dado el caso, deberá quedar como evidencia acta de reunión (Acta de entrega) y grabación del mismo

10.2 Pruebas unitarias:

Se comienza probando unidades individuales de código, como funciones o métodos. Se debe asegurar de que cada unidad funcione correctamente. Se Puede utilizar marcos de pruebas unitarias, como JUnit (para Java) o pytest (para Python).

El equipo de tester define las herramientas que se va a utilizar. Las pruebas realizadas deben quedar evidenciadas en el formato de pruebas unitarias GTIS01- F009 definidos por la Oficina Asesora de Informática.

10.3 Pruebas de integración:

Combina las unidades probadas en un sistema más grande y verifica que interactúen adecuadamente. Se debe asegurar de que los componentes se comuniquen correctamente y que los datos se transfieran sin problemas. Las pruebas realizadas deben quedar evidenciadas en el formato de pruebas de integración (Formato en construcción).

10.4 Pruebas de sistema:

Se encargan de probar todo el sistema en su conjunto para asegurarse de que cumpla con los requisitos especificados. Realizan pruebas de extremo a extremo para simular el flujo de trabajo completo del usuario. Las pruebas realizadas deben quedar evidenciadas en el formato de pruebas de integración (Formato en construcción).

10.5 Pruebas de aceptación:

Se llevan a cabo una serie de pruebas de aceptación para permitir que el cliente valide todos los requisitos. Estas pruebas tienen como objetivo validar el producto y verificar si este atiende a los requisitos especificados. Estas se realizan en un ambiente semejante al ambiente real de ejecución.

Durante estas pruebas y validación de aceptación, el equipo de desarrollo, clientes y los terceros se reúnen para discutir si se va a aceptar o no el producto. Si las pruebas son satisfactorias, se acepta el producto si no es satisfactorio se revisan y corrigen los problemas y se pasa a una segunda interación de este proceso de pruebas. Las pruebas realizadas deben quedar evidenciadas en el formato de pruebas de integración (Formato en construcción).

10.6 Pruebas de rendimiento y carga:

Las pruebas de carga forman parte de la garantía de calidad. Antes de lanzar un nuevo software, hay que comprobar qué requisitos necesita el programa y hasta dónde llega el límite de sus capacidades. Se comprueba cómo procesa el sistema una carga adicional inesperada, en qué momento se producen limitaciones y con qué rapidez reacciona ante los errores.

Las pruebas de carga también sirven para comprobar sistemas exigentes. Un ejemplo es el de las páginas web. Si los desarrolladores esperan una gran sobrecarga, pueden simularla con una prueba de carga y asegurarse así de que el sistema puede aguantar bien la carga adicional. Una sobrecarga puede producirse cuando se lanza un nuevo producto, se acerca un acontecimiento especial o la página web recibe muchas visitas debido a la publicidad. El objetivo de las pruebas de carga es comprobar la capacidad de carga de un sistema y su reacción, para poder mejorar de antemano el rendimiento cuando sea necesario. Las pruebas realizadas deben quedar evidenciadas en el formato de pruebas de integración (Formato en construcción).

Las solicitudes para el acceso a los servidores se deben realizar mediante el formato de control de accesos GTIGPS02-F005 ya definidos por la Oficina, las incidencias deben quedar evidenciadas en el formato de incidentes de seguridad ya definidos por la Oficina.

10.7 Pruebas de seguridad:

Responsable, área de seguridad OAI.

Responsabilidades del área de seguridad: Detectar, identificar, mitigar y realizar seguimiento.

Se debe realizar la evaluación de las potenciales vulnerabilidades. Las pruebas de seguridad del software aseguran que el producto final resulte seguro para los clientes, una vez finalice el desarrollo del software o aplicativo y antes de poner en producción se verifican si hay vulnerabilidades. Se debe llevar a cabo en análisis de riesgos para identificar las posibles amenazas y vulnerabilidades en el aplicativo. Estas permiten verificar si los diseños del sistema y del código puedan resistir a un ataque.

Las herramientas de seguridad permiten detectar las vulnerabilidades de seguridad y la posible explotación de éstas por parte de personas no autorizadas. Las pruebas de penetración se deben realizar para identificar posibles puntos de entrada para atacantes.

Herramientas:

Nessus essentials: (Se instala en una máquina virtual con sistema operativo Kali Linux). Es un escáner de vulnerabilidades que realiza pruebas para encontrar fallos de seguridad públicos. Estas pruebas se hacen a partir de bancos de información, que se actualizan constantemente y contienen datos sobre fallos de seguridad recientes. Este análisis de vulnerabilidades se hace a nivel de servidores, es decir, el área de desarrollo necesita un servidor para provisionar un servicio, por parte de seguridad se corre la herramienta sobre ese servidor y esta herramienta arroja las vulnerabilidades existentes dentro del servidor.
OWASP ZAP: Sobre el código fuente permite aplicar unos métodos de ataque, es decir, métodos de penetración, métodos de SQL inserción, si la aplicación tiene una base de datos el permite hacer la validación, a partir de esto la herramienta genera un informe donde se genera la línea de código que tiene el inconveniente y cuál es el problema y la solución, el informe presenta a manera de una guía práctica la solución ante ese problema o falla.

Al momento de publicar un sitio el área de seguridad se encarga de aplicar estas dos herramientas, los informes que se generan a nivel de servidor se utiliza Nessus essentials, a nivel de código fuente se utiliza OWASP ZAP.

Una vez se generen y finalicen estas pruebas los informes se envían directamente al área de desarrollo para la corrección de esos errores o vulnerabilidades, cuando el área de desarrollo haga las correcciones se vuelve a correr las herramientas cuantas veces sea necesario hasta reducir el mayor número de errores en el código y en sistema operativo.

Los informes generados mediante las herramientas indican si el sistema o aplicativo está libre de vulnerabilidades, si existen vulnerabilidades que no se pueden remediar se permite la aceptación de los riesgos por parte del dueño, responsable o administrador del sitio o sistema del uso de la aplicación (entidad, dependencia), serán ellos los responsables de asumir la publicación del sitio en esas condiciones.

10.8 Corrección de errores y mejoras:

Se debe corregir cualquier error identificado durante las pruebas. Se debe considerar realizar mejoras adicionales en función de la retroalimentación de los usuarios y las pruebas. Se debe relacionar los formatos ya definidos por la Oficina Asesora de Informática, informes, actas o cualquier otra evidencia necesaria. El gestor de desarrollo será la persona encargada de validar todos estos procesos. Se debe asegurar de que la documentación del software esté actualizada y sea comprensible para los usuarios y desarrolladores.

10.9 Lanzamiento:

El lanzamiento o paso a producción de un sistema o software se debe realizar después de que se finalice la etapa de aplicación de pruebas y correcciones, en donde se estructura un plan de despliegue especificado en el Formato de despliegue de software GTIGS01- F011, ya definido por la Oficina Asesora de Informática, donde se debe describir y completar la información solicitada.

El proceso de despliegue de un sistema o software de debe realizar en un periodo que no afecte al usuario, se debe planificar una actualización o migración de una plataforma o sitio cuando los usuarios no la estén usando, es decir, en horarios no laborales o de poco uso del sitio.

Se debe realizar un monitoreo continúo con el fin de analizar e identificar si hay una adecuada adaptación del software, tanto del personal que usa el aplicativo y también sobre el rendimiento del sistema, es decir, si éste se está ejecutando de manera correcta.

A medida que el software evoluciona se repite este ciclo de prueba en cada versión para garantizar su calidad continua.
Estas revisiones contantes se realizan con el acompañamiento del área de desarrollo, en donde se debe definir un plan de revisiones, es decir, se debe establecer unas fechas donde el equipo o área de desarrollo deberá evaluar.

En caso de que el aplicativo sea evaluado por el cliente, se procede a contactarlo, se le preguntará cómo está evolucionando el sistema, que fallas ha notado o evidenciado que presente el sistema, si presenta eficiencias en rendimiento y demás, se debe realizar con el fin de obtener un diagnóstico clave para poder verificar que el sistema se adecue a las necesidades y especificaciones del cliente. Si no se presentan anomalías se cierra la etapa de seguimiento. Si se presentan incidencias el cliente deberá remitirlas mediante correos, oficios, reuniones sesiones presenciales, el área de desarrollo deberá hacer el levantamiento de requerimientos para poder aplicarlos y de esta manera cumplir con lo solicitado, vuelve a iniciarse el ciclo nuevamente.

6. DOCUMENTOS DE REFERENCIA:

- Política de gobierno y seguridad digitales
- Documento de ambientes independientes en el ciclo de vida de los sistemas de información (1) GTIGS01-I002.

7. CONTROL DE CAMBIOS

VERSION	DESCRICPCION DE CAMBIOS
1.0	Elaboración del documento

8. VALIDACION DEL DOCUMENTO

ELABORADO POR:	REVISADO POR:	APROBADO POR:
Nombre: Deisy Johana Pechené Cargo: Documentadora software OAI Fecha: 26/09/23	Nombre: Jasmín Herrera – Anderson Pechené Cargo: Gestor Calidad – Gestor Desarrollo Fecha:26/09/23	Nombre: Ingrid Paola Solano Cargo: Jefe Oficina Asesora de Informática Fecha: 26/09/23