**Objetivo**:

Desarrollo de nuevas funcionalidades e integración de las misma en el sistema interno de trabajo de COSBIOME además de correcciones de errores, actualización de módulos o migración de algunos.

**Alcance:**

Este proceso abarca todas las áreas por el hecho de que cada departamento o la mayoría de una u otra forma usan el sistema interno o derivados de la empresa ya sea por medio de pc o celular en aplicaciones y cada departamento podría aportar una idea de mejora o detección de errores al momento de su uso.

**Políticas Generales:**

* Los lenguajes de desarrollo y control de las aplicaciones del sistema o derivados son:
  + DART (FLUTTER)
  + JavaScript (NODEJS, REACTJS, REACTNATIVEJS, STRAPIS, EXPRESSJS)
  + SQL (MySql community, MariaDB)
  + MongoDB (Atlas)
  + Python (TENSORFLOW, PYTORCH, SKLEARN, KERAS)
* Cada desarrollo que se haga con JavaScript ya sea de sistemas, herramientas de desarrollo, libreas internas o funciones de nube tendrán que ser desarrolladas con \*typescript e interfaces correspondientes.
* El \*lint del código se automatizará y verificará en cada desarrollo con las reglas \*Eslint-AIRBN&B y con el estándar de actualización de \*ECMASCRIP.
* El control de versiones de todos los desarrollos del sistema será llevado en Git en máquinas locales y en nube con GitHub para mantención de y copias de seguridad.
* Toda herramienta de desarrollo, librería o \*software para desarrollo que sea externo de COSBIOME tendrá que ser de \*software libre y con licencia MIT para la integración del mismo además de contar con repositorio público en GitHub y versiones \*LTS.
* Los ambientes de desarrollo para creación de nuevos módulos, medicaciones, actualizaciones o características para los desarrollos internos de COSBIOME tendrán que ser desarrollados en sistemas operativos basados en UNIX:
  + MacOS
  + Windows WSL
  + GNU & LIMUX
* \*WebPack será el estándar de compilación para cualquier desarrollo con JavaScript que necesite de la misma compilación.
* Conexión con servidores de producción remotos tendrán que ser por medio de llaves SSH y no se podrá habilitar contraseñas raíz por seguridad de la empresa.
* El CI/CD (INTEGRACION CONTINUNA/ DESPLIEGUE CONTINUO) se llevará en su totalidad por medio de:
  + Contenedores Docker
  + GitHub Acciones
  + \*Firebase herramientas
  + Jenkins
* Cada \*commit al repositorio principal tendrá que ser hecho \*fork clonado directamente del GitHub de COSBIOME a la cuenta de Ing. en sistemas en plantilla y con la nomenclatura:
  + FEAT: NUEVA FUNCIONALIDAD
  + FIX: CAMBIO DE ERRROR
  + UPD: PARA ACTUALIZACION DE LIBREARIA O FUNCIONES EN MODULO
  + CGN: PARA CAMBIOS MENORES COMO COLORES, ORTIGRAFIA ETC…
* Cada cambio en sistema y derivados tendrá que ser documentado con \*SketchBook dentro del código.
* Todos los desarrollos de pruebas ya sea FINAL A FINAL o UNITARIOS tendrán que ser realizados con:
  + \*Keras
  + \*TestingLibrary
  + \*JEST

|  |  |  |
| --- | --- | --- |
| **PROCESO** | **ACTIVIDAD** | **DESARROLLO** |
| 1. DETECCION DE PROBLEMA O PROCESO | * 1. Ideas de mejora o problemas a solucionar | 1.1.1. Es la aportación de los diferentes departamentos para agregar mejoras o corregir errores dentro del sistemas esto ayuda a tener un flujo de cambio y desarrollo dentro del sistema. |
| 1.2. Analís para el desarrollo y codificación de la nueva idea o problema a solucionar | 1.2.1. Una vez aportadas las ideas de cambio o lo reportes de errores es el momento de hacer en análisis para la estructurar el plan de codificación que se implementara para dar a la luz todas estas nuevas características. |
| 1.3. Plan de codificación para integraciones o correcciones en sistemas o derivados | 1.3.1. Es empezar a ver la detección del error donde se cambiarán las líneas de código o donde, como y con qué \*Apis se usarán los códigos, módulos o funciones para la generación de una nueva funcionalidad o modulo para crear la nueva aportación de mejora del departamento. |
| 1. CODIFICACION DE MODULO O FUNCIONALIODAD | * 1. Codificación de los nuevos módulos, funcionalidades o parches para corrección de errores | 2.1.1. Escribir ya el código como tal para creación de nuevos módulos, funcionalidades o corrección de los errores. |
| * 1. \*Lint de código para posteriormente general empaquetado del mismo | 2.2.2. Ajustarnos a las reglas de lint establecidas en las políticas para la posterior maquetación de modulo o función y a si llevar un orden de codificación en los diferentes desarrollos. |
| * 1. Empaquetado del módulo con WebPack | 2.3.1. Ejecutar comando (\*yarn build) de maquetación para tener un nuevo módulo, funcionalidad o corrección para generar las pruebas del mismo. |
| * 1. Codificar el test unitario para la integración del mismo con las librerías establecidas en las políticas a elección del desarrollador | 2.4.1 Aquí se escribe la prueba de integración que el módulo o función va a tener cada codificación que lleve un archivo extra dentro del sistema tendrá que tener su respectivo test de integración o su respectiva modificación si es que es un cambio por alguna detención de error. |
| 1. TESTO DE NUEVA FUNCION O MODULO | * 1. Ejecutar los testeos en archivos codificados | 3.1.1. Ejecutar el comando de testeos para el sistema (yarn test) que este arrojara si es que las pruebas de integración fueron realizadas. |
| * 1. Códigos de ejecución y \*logs de las pruebas de integración | 3.2.1. Salida de los códigos de ejecución de las pruebas realizadas además de los \*logs que nos darán la información de que si el módulo o funcionalidad paso las pruebas o no. |
| * 1. Verificación de códigos de ejecución y \*logs | 3.3.1. Ver los códigos y \*logs para poder ver si es que el test paso o no y si es esta última regresarlo a al proceso (2. CODIFICACION DE MODULO O FUNCIONALIODAD). |
| * 1. Prueba finalizada con códigos y \*logs aprobatorios para las pruebas de usuarios final a final | 3.4.1. Finalización de las pruebas con resultados positivos para su comprobación de las pruebas de usuarios final a final y pasa a ser manos de (ELIZABETH, ALEJANDRA Y MIA) para las pruebas. |
| 1. TESTO DE USUARIO FINAL A FINAL | * 1. Ejecutar prueba F2F con Elizabeth | 4.1.1. Elizabeth se encarga de que todos los componentes dentro de la interfaz gráfica funcionen además y que cada botón regrese un código de ejecución de las \*Apis en rango de 200-299 ya sea en http o de ejecución si es que no pasa se regresa al punto (2. CODIFICACION DE MODULO FUNCIONALIDAD) para cambiar la codificación para poder pasar la prueba. |
| * 1. Ejecutar prueba F2F con Alejandra | 4.2.1. Alejandra se encarga de que el código del nuevo módulo no se vea afecto por causas externas que provengan del navegador, dispositivo o entrono de uso del sistema por ejemplo problemas de internet lento, modificación en \*localstorage o \*sesionstorage o una modificación en el \*pront del HTML desde el inspector de tareas además de controlar las inyecciones de \*SQL que el entorno pueda sufrir por usuarios malicioso si es que no pasa se regresa al punto (2. CODIFICACION DE MODULO FUNCIONALIDAD) para cambiar la codificación para poder pasar la prueba. |
| * 1. Ejecutar prueba F2F con MIA | 4.3.1. Mia se encarga de fungir como usuario malicioso atacando los procesos de las pruebas de Elizabeth y Alejandra para generar un entorno de riesgo en simulación para asegurar la robustez de las pruebas y seguridad dentro del sistemas si es que Mia logra romper alguna prueba de alguna de las dos se regresa al punto (2. CODIFICACION DE MODULO FUNCIONALIDAD) para cambiar la codificación para poder pasar la prueba. |
| * 1. Modulo testeado con entornos de ejecución casi similares a los de producción | 4.4.1. Por último, tenemos el módulo ya testeado para su maquetación y liberación de en el servidor de producción. |
| 1. DESPLIEGUE A PRODUCCION DE NUEVA FUNCIONALIDAD O MDULO | * 1. Despliegue a producción de nuevo módulo, función corrección de error | 5.1.1. Ejecutar el script de ejecución de despliegue para hacer la imagen del sistema con Docker y el despliegue a producción ene l servicio de \*Firebase el script ya viene con el repositorio del código \*(bash ./deploy.sh). |
| * 1. Comprobación de redundancias en actualizaciones de despliegue | 5.2.1. Verificar que alguna actualización de sistemas anterior o posterior no tenga código viejo que pueda afectar a una actualización mayor y que haga que producción tenga una versión anterior si es así la versión se cancela y da paso a la nueva. |
| * 1. Notificar a usuarios del sistema activos que hay una nueva actualización | 5.3.1. Notificar por medio del sistema a los usuarios activos conectados que una nueva actualización ha sido liberada para que puedan actualizar el sistema por medio de la instrucción de dar control + R. |
| 1. USO DE NUEVAS FUNCIONES (USUARIOS) | * 1. Confirmar con usuarios las nuevas funciones | 6.1.1. Confirmar con los usuarios que las nuevas funciones de sistemas o módulos las estén usando para la posterior recolección de datos para encontrar errores o no descubiertos en las pruebas o pensar en nuevas funciones que se pudieran agregar a futuro como módulos o funcionalidades. |

**ANEXOS/DOCUMENTOS**

*TODAS LAS PALABRAS QUE INCIEN CON UN (\*) SON PALABRAS EN INGLES QUE NO TIENEN UN TRADUCCION LITERAL AL ESPAÑOL QUE LA COMUNIDAD DE IT A NIVEL GLOBAL CREARON PARA DESAROLLOS EN PROGRAMACION.*

*SIEMPRE CAMBIOS EN ACTUALIZACIONES SE TIENEN QUE AGREGAR A LA DOCUMENTACION UNA VEZ QUE PASEN LAS PRUEBAS FNF Y SE PUEDAN LANZAR A PRODUCCION.*

*LOS NOMBRES “FENINOS” QUE SE MENCIONAN EN LOS PUNTOS 3 Y 4 SON NOMBRES DE INTELIGENCIAS ARTIFIACLES LAS CUALES AYUDAN CON LOS PROCESOS DE PRUEBAS DENTRO DEL SISTEMA.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Control de Emisión** | | | |
|  | **Puesto** | **Nombre** | **Fecha** |
| **Elaboró** | ENCARGADO DE SISTEMAS | Rizo Quintero Juan Pablo | 28/11/2022 |
| **Revisó** |  |  |  |
| **Autorizó** |  |  |  |