**MysticalCut**

**Plan de Migración**

Versión: 0001

Fecha: 8/07/2025

**HOJA DE CONTROL**

|  |  |  |  |
| --- | --- | --- | --- |
| **Organismo** | SENA | | |
| **Proyecto** | MYSTICALCUT | | |
| **Entregable** | Plan De Migracion | | |
| **Autor** | THE BROTHER | | |
| **Versión/Edición** | 0001 | **Fecha Versión** | 08/07/2025 |
| **Aprobado por** |  | **Fecha Aprobación** | DD/MM/AAAA |

REGISTRO DE CAMBIOS

|  |  |  |  |
| --- | --- | --- | --- |
| **Versión** | **Causa del Cambio** | **Responsable del Cambio** | **Fecha del Cambio** |
| 0001 | Versión inicial | Harold David Hernandez, Oscar Andres Galarza, Andres Esteban Castañeda, Kevin David Sabogal | 08/07/2025 |
|  |  |  |  |
|  |  |  |  |

CONTROL DE DISTRIBUCIÓN

|  |
| --- |
| **Nombre y Apellidos** |
| Harold David Hernández Vásquez |
| Oscar Andres Leon Galarza |
|  |
|  |
|  |

## 

## 

## **1.objetivo**

El propósito fundamental de este plan es guiar de manera segura y controlada el proceso de migración del sistema **MysticalCut**, abarcando sus componentes **web**. Este enfoque asegura dos pilares muy importantes dentro de nuestro sistema:

* **Integridad y Respaldo de Datos:** Garantizar que, durante y después de la migración, todos los datos del sistema permanezcan intactos, precisos y accesibles. Esto incluye la implementación de estrategias robustas de respaldo y verificación para prevenir cualquier pérdida o corrupción de información.
* **Minimización de Riesgos:** Identificar proactivamente, evaluar y mitigar los posibles riesgos asociados con el proceso de migración. Esto busca reducir al máximo las interrupciones del servicio, los fallos operativos y cualquier impacto negativo en la funcionalidad o el rendimiento del sistema.

## **2.Alcance**

Este plan de migración se enfoca específicamente en el **Backend** del sistema **MysticalCut.** La migración comprenderá los siguientes componentes tecnológicos:

* **Tecnología Principal:** El backend está desarrollado utilizando **Spring Boot con Kotlin**.
* **Base de Datos:** La base de datos asociada a este backend es **MySQL**.

El plan detallará los pasos y consideraciones necesarios para migrar de forma efectiva estos componentes, asegurando su correcto funcionamiento y compatibilidad en el nuevo entorno o con las versiones actualizadas.

## **3.Consideraciones Generales**

Antes de iniciar cualquier proceso de migración en el sistema **MysticalCut**, es imprescindible realizar una evaluación integral que permita visualizar de forma clara el contexto del proyecto, los objetivos perseguidos y los posibles desafíos técnicos y funcionales. Esta fase preliminar garantiza que la migración se ejecute sobre una base sólida, minimizando riesgos y facilitando la toma de decisiones acertadas.

#### **Justificación de la Migración (Técnica y Funcional)**

**¿Por qué se hace la migración?**

Es necesario definir con precisión las motivaciones que justifican este cambio, desde dos enfoques complementarios:

**Técnica**, la migración puede responder a necesidades como:

* Eliminar dependencia de tecnologías obsoletas o no soportadas.
* Mejorar el rendimiento del sistema, optimizando tiempos de respuesta y uso de recursos.
* Incrementar la seguridad mediante la adopción de versiones actualizadas con parches críticos.
* Escalar la solución para soportar una mayor carga o integración con nuevos servicios.
* Facilitar el mantenimiento y la incorporación de buenas prácticas modernas, como la adopción de Kotlin o frameworks más recientes como Spring Boot 3.

**Funcional**, la migración busca:

* Potenciar la experiencia del usuario con interfaces más fluidas y confiables.
* Permitir la implementación de nuevas funcionalidades antes limitadas por la arquitectura o tecnología actual.
* Aumentar la disponibilidad del sistema reduciendo el tiempo de inactividad.
* Fortalecer la estabilidad y previsibilidad del comportamiento del sistema en producción.

#### **Impacto en Módulos y Funcionalidades Existentes**

**¿Qué se verá afectado?**

Es indispensable realizar un mapeo detallado de todos los módulos del sistema para identificar cómo se verán impactados por la migración. Este análisis debe considerar:

* **Dependencias técnicas**: bibliotecas, frameworks y configuraciones que deben actualizarse.
* **Interacciones entre módulos**: conexiones o flujos de datos que podrían romperse temporalmente si no se sincroniza correctamente el proceso.
* **Modificaciones necesarias en el código fuente**: especialmente en capas como controladores, servicios, repositorios y entidades.
* **Riesgos sobre funcionalidades críticas**: aquellas que están en uso constante por parte de los usuarios, como el agendamiento de citas, la gestión de productos, ventas o generación de reportes.
* **Impacto en pruebas**: necesidad de adaptar o reescribir casos de prueba automatizados o manuales.
* **Tiempos de indisponibilidad estimados**: planificación para minimizar afectaciones al entorno productivo.

# **4. Respaldos y Preparación**

La disponibilidad y la integridad de los datos son críticas en cualquier migración. Para MysticalCut, este punto es fundamental.

### **Base de Datos:** Vamos a realizar un dump completo de la base de datos de producción MySQL. Nuestro "Plan de Respaldo" debe ser específico:

* **Herramienta a utilizar**: Emplearemos mysqldump para garantizar una copia fiel y completa de nuestra base de datos.
* **Frecuencia:** Para entornos intermedios (desarrollo, staging), se realizará un respaldo antes de cualquier cambio significativo. Para producción, el respaldo completo se hará inmediatamente antes de la ventana de migración.
* **Ubicación de almacenamiento:** Los respaldos se guardarán en una ubicación de almacenamiento segura y accesible, fuera del servidor de producción, como un almacenamiento en la nube dedicado o un servidor de respaldo.

### **Archivos Críticos:** Además de la base de datos, es vital respaldar todos los archivos críticos del sistema MysticalCut:

* **Archivos de configuración:** Incluiremos copias de .env, application.properties, y cualquier otro archivo de configuración específico del entorno de Spring Boot y Kotlin.
* **Recursos estáticos:** Aseguraremos copias de seguridad de todas las imágenes, hojas de estilo CSS, archivos JavaScript y cualquier otro recurso estático que use la aplicación.
* **Certificados SSL:** Es crucial respaldar los certificados SSL y sus claves privadas asociadas para asegurar la conectividad segura post-migración.

# **5.** **Metodología y Objetivos**

Nuestra metodología se centrará en la seguridad, la automatización y la verificación constante.

* **Respaldo Completo de Datos:** El objetivo primordial es que los datos estén respaldados de forma completa y verificada antes de realizar cualquier cambio significativo en el sistema.
* **Integración y Entrega Continua (CI/CD):** Utilizar CI/CD para **validar los cambios** de manera continua. Esto significa que cada cambio en el código pasará por un codigo automatizado de pruebas y despliegue para detectar problemas tempranamente.
* **Documentación de Cambios:** **Documentar exhaustivamente todos los cambios realizados** durante el proceso de migración. Esto incluye modificaciones de código, configuraciones, dependencias, y cualquier paso ejecutado.
* **Despliegue por Etapas:** Priorizar el **despliegue en entornos de prueba (staging)** antes de pasar a producción. Esto permite validar la funcionalidad y el rendimiento en un entorno que simula el de producción sin afectar a los usuarios finales.

## **6. Equipos Relacionados**

La colaboración entre equipos es clave para el éxito de la migración.

* **Equipo de Desarrollo y Migración (EDM):** Responsable de la ejecución técnica de la migración, incluyendo el desarrollo de código, la configuración de entornos, y la resolución de problemas técnicos.

**Integrantes:** Kevin David Sabogal

Andrés Esteban Castañeda

Oscar Andrés León

Harold David Hernández

* **Responsables de Control de Calidad (QA):** Encargado de definir y ejecutar los planes de pruebas, identificar defectos, y asegurar que la funcionalidad del sistema se mantenga intacta después de la migración.

**Integrantes:** Harold David Hernández Vásquez

### **7. Estrategia Técnica: Rama de Migración y Entrega Continua**

Para garantizar una migración limpia y sin afectar la rama principal de desarrollo, se seguirá una estrategia de ramificación y un flujo de trabajo de entrega continua.

### **Uso de Rama Secundaria (migracion-x)**

* **Creación de Rama Específica:** Se creará una rama específica para la migración, por ejemplo, migracion-v2. Esta rama será el entorno de trabajo principal para todos los cambios relacionados con la migración.
* **Centralización de Cambios:** Todo cambio estructural, de versiones, o de dependencias (por ejemplo, actualizaciones de Spring Boot o versiones de Kotlin) se realizará **exclusivamente en esta rama**.
* **Flujo de Pull Request (PR):** Las actualizaciones y nuevos desarrollos en la rama de migración deben seguir un flujo de **Pull Request con revisión por pares**. Esto asegura la calidad del código, la adhesión a los estándares y la detección temprana de posibles problemas.
* **Aislamiento de la Rama Principal:** Se asegura que la rama principal (main o production) **no se vea afectada** por los cambios de migración hasta que la rama de migración haya sido completamente validada y aprobada.

**Validación de Migración**

* Una vez que los cambios en la rama de migración han sido implementados y probados internamente, se procederá a la validación.
* **Despliegue en Staging:** La rama de migración será desplegada en el entorno de staging para pruebas exhaustivas por parte del equipo de QA y, si aplica, por usuarios clave.
* **Pruebas de Integración y Rendimiento:** Se ejecutarán pruebas de integración para asegurar que todos los componentes se comuniquen correctamente y pruebas de rendimiento para verificar que el sistema mantiene o mejora su eficiencia.

### **Merge Controlado**

* Después de una validación exitosa en staging y la aprobación por parte de todos los equipos involucrados, se procederá con el merge de la rama de migración a la rama principal.
* **Programación:** El merge se realizará en un período de bajo tráfico para minimizar el impacto en los usuarios.Inmediatamente después del merge, se establecerá un monitoreo intensivo para detectar cualquier anomalía o problem

**8. Gestión de Riesgos**

|  |  |  |
| --- | --- | --- |
| **Riesgo** | **Impacto** | **Mitigación** |
| Incompatibilidad con nuevas versiones | Alto | Realizar **pruebas exhaustivas en el entorno de staging** antes de la producción. **Documentar todos los *breaking changes*** de las nuevas versiones de las tecnologías (Spring Boot, Kotlin, librerías) y ajustar el código en consecuencia. Mantener un registro de versiones. |
| Pérdida o corrupción de datos | Alto | **Respaldos completos y verificados** de la base de datos y archivos críticos antes de cualquier cambio. Plan de rollback documentado. |
| Problemas de rendimiento | Medio | Realizar **pruebas de carga y estrés** en el entorno de staging. Monitoreo constante del rendimiento post-migración. |
| Fallos en integraciones externas | Medio | Identificar todas las **integraciones con sistemas de terceros** y probarlas exhaustivamente en el entorno de staging. Contactar a proveedores si es necesario. |
| Interrupción prolongada del servicio | Alto | Planificar la migración en **ventanas de mantenimiento programadas**. Implementar una estrategia de rollback clara y probada. Comunicación transparente a los usuarios sobre posibles interrupciones. |
| Regresión de funcionalidades | Medio | **Ejecución de un conjunto completo de pruebas de regresión** (manuales y automatizadas) después de la migración. |
| Falta de conocimiento técnico | Bajo | Asegurar que el equipo tenga la **capacitación adecuada** en las nuevas versiones o tecnologías. Compartir conocimiento y documentación interna. |

### **9. Plan de Validación y Pruebas (con Automatización)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo de Prueba** | **Descripción** | **¿Automatización?** | **Herramientas / Enfoque Utilizado** |
| **Pruebas unitarias** | Verifican que cada parte del código (por ejemplo, funciones o métodos) funcione de forma correcta. | Sí | Herramientas de pruebas integradas en el entorno de desarrollo. Se ejecutan automáticamente al guardar cambios. |
| **Pruebas de integración** | Comprueban que los diferentes módulos del sistema se comuniquen correctamente entre sí y con la base de datos. | Sí | Se utilizarán entornos de prueba controlados para asegurar que todos los componentes trabajen bien juntos. |
| **Pruebas funcionales (End-to-End)** | Validan el sistema completo como lo usaría un usuario real, probando procesos clave como reservas, pagos, etc. | Sí | Se simulará el uso del sistema, generando reportes con los pasos realizados. Se evaluarán tanto la versión web |
| **Pruebas de rendimiento y carga** | Evalúan cómo responde el sistema con muchos usuarios al mismo tiempo, o ante tareas pesadas. | Sí | Se harán simulaciones de múltiples usuarios usando el sistema para medir velocidad, estabilidad y capacidad. |
| **Pruebas de regresión** | Aseguran que nuevas actualizaciones no dañen funciones que antes trabajaban bien. | Sí | Se ejecutará un conjunto de pruebas ya definidas cada vez que se haga un cambio importante. |
| **Pruebas de seguridad** | Detectan posibles fallos que puedan poner en riesgo la información del sistema o de los usuarios. | Parcialmente | Se realizarán revisiones automáticas del código y pruebas manuales para encontrar vulnerabilidades comunes. |

**Automatización con Cucumber + Serenity**

La combinación de **Cucumber y Serenity** permite automatizar pruebas funcionales de forma clara y comprensible tanto para desarrolladores como para usuarios no técnicos.

#### **Cualidades principales de la herramienta:**

* **Lenguaje natural:** Permite escribir los escenarios de prueba en lenguaje sencillo (Gherkin), facilitando la colaboración entre desarrolladores, testers y analistas.
* **Reportes visuales:** Serenity genera informes detallados y visuales que incluyen:
  + Los pasos ejecutados en cada prueba.
  + Evidencias como capturas de pantalla (cuando aplica).
  + Estado de cada paso (éxito o fallo).
  + Seguimiento del avance de pruebas y cobertura funcional.
* **Facilita el análisis de fallos:** En caso de errores, los reportes ayudan a identificar rápidamente en qué parte del proceso falló la prueba.
* **Reutilización de código:** Serenity permite estructurar bien las pruebas, promoviendo la reutilización de pasos comunes, lo que reduce el esfuerzo de mantenimiento.

### **10. Documentación y Control de Versiones**

* Una documentación exhaustiva y un sistema de control de versiones bien estructurado son esenciales para garantizar la trazabilidad, facilitar la auditoría técnica y preservar el conocimiento a lo largo del proceso de migración.

#### **Documentación Pre-Migración**

* **Arquitectura Actual del Sistema:**  
   Elaborar una documentación completa de la arquitectura vigente, incluyendo:
  + Diagramas de componentes (módulos, capas, dependencias).
  + Flujos de datos y procesos.
  + Interfaces entre servicios y sistemas externos.
* **Inventario de Versiones:**  
   Registrar las versiones específicas de:
  + Frameworks y librerías utilizadas.
  + Base de datos y motor de ejecución.
  + Lenguajes de programación y herramientas asociadas.
* **Checklist de Respaldo:**  
   Definir y documentar un checklist detallado con todos los elementos que deben ser respaldados antes del inicio de la migración, incluyendo:
  + Bases de datos.
  + Archivos de configuración.
  + Código fuente.
  + Logs relevantes.
  + Recursos estáticos y archivos adjuntos.

#### **Control de Versiones (Git)**

* **Sistema de Control de Versiones:**  
   Utilizar **Git** como sistema centralizado de control de versiones para gestionar el código fuente y los artefactos de configuración durante la migración.
* **Ramas Estratégicas:**  
   Definir y mantener las siguientes ramas con propósitos claros:
  + **main:** Versión estable y lista para producción.
  + **main:** Versión en desarrollo activo.
  + **main**: Versión actualmente desplegada en el entorno productivo.
  + **migracion-x:** Rama específica para gestionar los cambios relacionados con la migración.
* **Historial de Cambios:**  
   Asegurar que cada commit:
  + Sea atómico y descriptivo.
  + Incluya referencias a tareas o tickets asociados.
  + Documente claramente los cambios realizados, su propósito y posible impacto.
* **Etiquetado Semántico:**  
   Utilizar **tags** en Git para marcar versiones clave del proyecto, como:
  + v1.0-pre-migracion: Versión estable antes de iniciar la migración.
  + v2.0-post-migracion: Versión estable después de completar la migración.
  + Otros hitos relevantes del proceso.

#### **Documentación Post-Migración**

* **Actualización de la Documentación Técnica:**  
   Inmediatamente después de la migración, se debe:
  + Actualizar la documentación de la arquitectura del sistema.
  + Registrar las nuevas versiones de componentes y tecnologías utilizadas.
  + Incorporar lecciones aprendidas y posibles incidencias detectadas durante la migración.

### **11. Capacitación y Comunicación**

* **11. Comunicación y Capacitación**

Una comunicación efectiva y una capacitación bien planificada son esenciales para garantizar una transición fluida, minimizar la resistencia al cambio y asegurar que tanto los usuarios finales como los equipos técnicos puedan adaptarse rápidamente al nuevo entorno.

#### **Capacitación a Usuarios Finales**

* **Talleres Interactivos o Videos Instructivos:**
  + Organizar sesiones prácticas o crear cápsulas en video de corta duración que expliquen de manera clara los cambios introducidos en:
    - La interfaz de usuario.
    - Nuevas funcionalidades.
    - Modificaciones en los flujos operativos.
* **Manuales de Usuario Actualizados:**
  + Proporcionar documentación sencilla y visual que refleje fielmente el sistema post-migración.
  + Incluir secciones de preguntas frecuentes (FAQ) y pasos para resolver errores comunes.
* **Evaluación y Retroalimentación:**
  + Aplicar ejercicios prácticos o encuestas breves para:
    - Verificar la comprensión del nuevo sistema.
    - Recolectar sugerencias o incidencias desde la perspectiva del usuario.

#### **Capacitación a Equipos Técnicos**

* **Sesiones Técnicas de Capacitación:**
  + Realizar entrenamientos técnicos sobre:
    - Nuevas versiones de frameworks y librerías.
    - Cambios en APIs internas o externas.
    - Procedimientos de despliegue (incluyendo automatización con CI/CD).
    - Uso de nuevas herramientas de monitoreo, testing o documentación.
* **Procedimientos de Rollback:**
  + Asegurar que todos los miembros técnicos estén familiarizados con:
    - Los pasos detallados para revertir la migración.
    - Las condiciones que activarían un rollback.
    - El plan de contingencia en caso de falla.
* **Manuales Técnicos y de Mantenimiento:**
  + Documentar la nueva arquitectura técnica, configuraciones del entorno, scripts de despliegue, puntos de monitoreo, logs relevantes, y tareas de mantenimiento recurrentes.
* **Canal de Comunicación Dedicado:**
  + Habilitar un canal centralizado de comunicación técnica (como Slack, Microsoft Teams, Discord, etc.) que funcione como soporte durante y después de la migración para:
    - Resolver incidencias en tiempo real.
    - Compartir soluciones a problemas comunes.
    - Mantener informados a todos los actores involucrados.

### **12. Post-migración**

La fase de post-migración es crítica para consolidar el éxito del proceso, garantizar la estabilidad operativa del sistema y resolver rápidamente cualquier incidente residual. Durante esta etapa, se prioriza el monitoreo, la validación de datos y la retroalimentación de los usuarios.

#### **12.1 Monitoreo Intensivo (Primeras 24-72 Horas)**

Se establecerá una ventana de observación continua tras la migración con el objetivo de detectar tempranamente cualquier comportamiento anómalo o degradación del servicio. Este monitoreo incluirá:

* **Rendimiento del Sistema:**
  + Uso de CPU y memoria en servidores.
  + Latencia y tiempo de respuesta de APIs.
  + Rendimiento de consultas y latencia en base de datos.
* **Errores de Aplicación:**
  + Captura y registro de excepciones.
  + Seguimiento de errores HTTP (404, 500, etc.).
  + Notificación de errores funcionales o lógicos.
* **Análisis de Logs:**
  + Supervisión en tiempo real de los logs de sistema y aplicación.
  + Identificación de patrones irregulares o fallas repetitivas.
  + Uso de herramientas centralizadas como ELK Stack, Grafana Loki o Splunk.

#### **Gestión de Errores e Incidentes**

* **Registro y Priorización:**
  + Todo error o comportamiento inesperado será documentado en un sistema de gestión de incidencias (como Jira, Trello o Redmine).
  + Las incidencias se clasificarán por nivel de criticidad y se priorizará su resolución.
* **Proceso de Escalamiento:**
  + Definir rutas claras de comunicación y niveles de escalamiento para la resolución oportuna de problemas críticos.

#### **Validación de Integridad de Datos**

* **Verificación Comparativa:**
  + Comparación entre datos previos y posteriores a la migración:
    - Conteo de registros por entidad.
    - Validación de claves primarias/foráneas.
    - Comprobación de checksums o hashes si aplica.
* **Auditoría Selectiva:**
  + Revisión de registros clave y transacciones sensibles para asegurar que no hubo pérdida, duplicación ni corrupción de datos.

#### **Feedback de Usuarios**

* **Recolección Activa:**
  + Habilitar formularios breves o canales directos (correo, chat, ticket) para que los usuarios puedan reportar:
    - Fallos funcionales.
    - Comportamientos inesperados.
    - Sugerencias de mejora.
* **Análisis del Feedback:**
  + Clasificar los reportes recibidos.
  + Priorizar los hallazgos relevantes que no hayan sido detectados durante las pruebas previas.