ParKeatec

Plan de Migración

Versión: 0100

Fecha: 08/07/2025

HOJA DE CONTROL

Organismo	Sena		
Proyecto	ParKeatec		
Entregable	Plan de migración		
Autor	Sena		
Aprobado por		**Fecha Aprobación**	DD/MM/AAAA
		Nº Total de Páginas	10

REGISTRO DE CAMBIOS

Versión	Causa del Cambio	Responsable del Cambio	Fecha del Cambio
0100	Versión inicial	Felipe Lugo	08/07/2025

CONTROL DE DISTRIBUCIÓN

Nombre y Apellidos
Felipe Lugo

Contenido

1 Objetivo

Establecer una estrategia integral para llevar una migración del sistema de manera planificada, segura y controlada, garantizando la continuidad del servicio, la integridad de la información, asegurando:

- Integridad y respaldo de datos.
- Minimización de riesgos.
- Compatibilidad con el entorno.
- Pruebas de funcionalidad y rendimiento.
- Documentación del proceso y plan de contingencia.

2 Alcance

Este plan cubre la migración de:

- Backend: Node.js.
- Frontend Web: React.js
- Frontend Móvil: React Native.

3 Consideraciones Generales

Antes de comenzar, evalúa:

¿Por qué se hace la migración?

La migración se realiza para modernizar la arquitectura del sistema, mejorar las escalabilidad y mantenimiento, y permite una integración más eficiente. Teniendo en cuenta las limitaciones técnicas, tiempos de respuesta y la dificultad para incorporar nuevas funcionalidades para los usuarios.

Impacto en módulos y funcionalidades existentes.

- Módulo de autenticación y seguridad
    - Cambios en la forma de gestión de token, sesión o roles.
    - Riesgos de compatibilidad si se cambia el sistema de login o la autenticación.
- Módulo de reservas o solicitudes
    - Cambios en la estructura lógica de datos pueden afectar como se guardan o procesan las reservas.
    - Requiere pruebas para evitar perdidas de datos o mal funcionamiento.
- Módulo de reportes
    - Puede requerir adaptaciones si se cambia el formato de datos o la tecnología usada para generarlos.
    - Podría presentar fallas temporales al no tener una reconfiguración correcta.

4. Respaldos y Preparación

- **Base de datos**: realizar un *dump completo*(Plan de respaldo).
- **Archivos críticos**: respaldar archivos de configuración (. env, application.properties, etc.)
- **Recursos estáticos y del sistema:**
    - Imágenes, logos, plantillas PDF o JS/CSS estáticos.
    - Carpetas /assest, /uploads, /public o similares.

- Documentación del sistema
    - Manuales técnicos
    - Diagramas de arquitectura
    - Documentación del código o de API.

- **Evidencia**: registrar fecha y ubicación de los respaldos.
- **Verificación de integridad**: Comprobar que los respaldos se puedan restaurar correctamente.
- **Detección de dependencias**:
    - Revisar versiones de librerías/frameworks (Node, React.js)
    - Asegurar la compatibilidad en el nuevo entorno (Java version, Node.js)
- Revisión de recursos del nuevo entorno: Espacio suficiente de disco duro o dispositivo de respaldo.

5. Metodología y Objetivos

- Respalda datos de forma completa antes de cualquier cambio.
- Usa integración y entrega continua para validar cambios.
- Documenta todos los cambios realizados.
- Despliega primero en entornos de prueba (staging) antes de producción.
- Define un cronograma por fases y responsables.
- Aplicar pruebas automatizadas y manuales.
- Usa versionamiento del código y las configuraciones.
- Mantener una comunicación activa durante todo el proceso.
- Tener listo un plan de reversión ante errores graves.
- Evaluar el proceso al finalizar y documenta novedades o lecciones aprendidas.

6. Equipos Relacionados

- Equipo de Desarrollo y Migración (EDM)
- Responsables de Control de Calidad (QA)
- Equipo de infraestructura/DevOps
- Líder de proyecto.
- Soporte técnico.

7. Estrategia Técnica: Rama de Migración y Entrega Continua

Para garantizar una migración limpia y sin afectar la rama principal:

- Uso de Rama Secundaria (migracion-x)
- Se crea una rama específica para la migración (migracion-Parkeatec-v2).
- Todo cambio estructural, de versiones o dependencias se realiza solo en esta rama.
- Las actualizaciones deben seguir el flujo de Pull Request con revisión por pares.
- Se asegura que la rama principal (main o production) no se vea afectada hasta la aprobación.
- Integración de CI/CD se ejecutan pruebas automáticas (unitarias, de integración, linting) mediante GitHub Actions, GitLab CI o Jenkins."
- Testing por módulos post-despliegue
- Trazabilidad con tickets y validación por checklist QA:

Validación de Migración

- Pruebas de funcionalidades
- Pruebas de seguridad
- Comparaciones con la versión actual y la migración para evitar errores y perdida de datos.
- Pruebas de rendimiento.

Merge Controlado

- Pull Request revisado y aprobado por el equipo.
- Checklist de validación cumplido (QA y funcional).
- Despliegue inicial en staging o pre-producción.
- Aprobación del líder del proyecto o del equipo técnico.
- Despliegue en producción acompañado de monitoreo.

8. Gestión de Riesgos

Riesgo	Impacto	Mitigación
Incompatibilidad con nuevas versiones	Alto	Pruebas en entorno de staging, documentación de breaking changes
Perdida de datos	Alto	Probar la migración en un entorno de pruebas validando la integridad de los datos.
Problemas de rendimiento	Medio	Implementar monitoreo Post-migración
Falta de capacitación	Medio	Manuales y una capacitación previa a la migración
Vulnerabilidades de seguridad	Alto	• Revisar permisos y roles • Usar conexiones seguras (HTTPS, SSL, TLS) • Hacer pruebas de seguridad

9. Plan de Validación y Pruebas (con Automatización)

Tipo	Descripción	Automatización	Herramientas
Pruebas unitarias	Validan métodos aislados	No	Pruebas en desarrollo
Prueba del sistema	Se verifica que todos los componentes funcionen en conjunto	Si	Cucumber + Serenity, Java, Gradle
Pruebas de regresión	Validar que todo funcione correctamente ante un cambio.	No	Jira, Excel
Prueba de aceptación	Que el software cumpla los requisitos del cliente o usuario.	No	PDF, Manuales y Reportes

Automatización con Cucumber + Serenity
Cualidades de la herramienta:

- Serenity genera reportes visuales detallados, incluyendo pasos, capturas de pantalla y evidencias.
- Gradle sirve para **compilar, construir, probar y empaquetar aplicaciones** de forma eficiente y automatizada.

Herramientas manuales

Cualidades de la herramienta:

- Prueba de desarrollo: Permite detectar errores de forma temprana al verificar pequeñas unidades del código, asegurando que los componentes se comporten según lo esperado, lo que contribuye a reducir costos y tiempos de desarrollo.

- Jira: Herramienta de gestión de proyectos que proporciona un tablero virtual para la creación de tareas, asignación a miembros del equipo, incorporación de listas de verificación, establecimiento de fechas límite y seguimiento del progreso de cada actividad.

10. Documentación y Control de Versiones

- Documenta arquitectura actual, versiones, checklist de respaldo antes de migrar.
- Controla versiones con Git.
- Ramas principales
- Bitácora de errores e incidencias durante el proceso.
- Asignar responsable para cada parte de la documentación.
- Registro de cambios por versión.

11. Capacitación y Comunicación

Usuarios finales:

- Talleres o videos cortos para mostrar cambios.
- Manuales actualizados.
- Evaluación con ejercicios o encuestas.
- Comunicación constante con el capacitador.
- Demostraciones en vivo

Equipos técnicos:

- Sesiones sobre nuevas versiones, APIs, procedimientos de rollback, uso de CI/CD.
- Manuales técnicos.
- Canal de comunicación para dudas.
- Simulacro de errores y recuperaciones
- Bitácora de conocimientos.

12. Post-migración

- Monitoreo intensivo durante las primeras 24-72 horas.
- Registro de errores o inconsistencias.
- Realización de backups posteriores como punto de restauración estable.
- Elaboración de informe de cierre con lecciones aprendidas.