Manifiesto de Codificación: Guía para un Desarrollo Ágil y Eficaz

En el dinámico mundo del desarrollo de software, la calidad y la eficiencia son pilares fundamentales para el éxito de cualquier proyecto. Es por ello que el Manifiesto de Codificación surge como una guía invaluable, estableciendo principios y buenas prácticas que permiten a los desarrolladores crear aplicaciones robustas, escalables y mantenibles.

Principios

1. La simplicidad como brújula:

Evite la complejidad innecesaria en su código. Opte por soluciones simples, directas y fáciles de entender. Recuerde: “Lo que se puede expresar de manera sencilla, no se debe complicar”.

2. Priorice la legibilidad:

Escriba código claro y conciso que se explique por sí mismo. Utilice nombres de variables descriptivos, comentarios relevantes y una estructura organizada. Un código legible facilita la comprensión, el mantenimiento y la colaboración.

3. Pruebas unitarias: la base de la confianza:

Implemente pruebas unitarias exhaustivas para cada componente de su código. Estas pruebas automatizadas le brindarán la seguridad de que cada módulo funciona correctamente, aislando errores y previniendo regresiones.

4. CI/CD: automatización a su servicio:

Adopte prácticas de Integración Continua (CI) y Entrega Continua (CD) para automatizar el proceso de construcción, pruebas y despliegue de su software. Esto agiliza el desarrollo, reduce errores y garantiza una entrega constante de valor al usuario.

5. Documentación: la memoria del código:

Documente su código de manera completa y precisa. Explique el propósito de cada módulo, las funciones de cada clase y las decisiones de diseño tomadas. Una documentación clara facilita la comprensión del código para otros desarrolladores y futuros colaboradores.

6. Despliegue controlado:

Implemente estrategias de despliegue controladas que minimicen el riesgo de errores en producción. Utilice entornos de preproducción para realizar pruebas exhaustivas antes de liberar el software a los usuarios finales.

7. La refactorización: un arte para mejorar:

No tema refactorizar su código para mejorar su estructura, legibilidad y mantenibilidad. La refactorización continua le permite mantener un código limpio y eficiente a lo largo del tiempo.

8. El código limpio es un código feliz:

Establezca estándares de codificación claros y consistentes dentro de su equipo. Utilice herramientas de análisis de código estático para identificar y corregir posibles errores de sintaxis, estilo y seguridad.

9. La colaboración: la clave del éxito:

Fomente la colaboración y el intercambio de conocimientos entre los miembros del equipo. Realice revisiones de código periódicas para identificar áreas de mejora y aprender unos de otros.

10. La retroalimentación: un regalo invaluable:

Busque y reciba retroalimentación constante sobre su código. Escuche atentamente las sugerencias de sus compañeros, usuarios y clientes. La retroalimentación le permite identificar oportunidades de mejora y adaptar su código a las necesidades reales.

Manifiesto de Políticas y Buenas Prácticas para Desarrollo de Microservicios

Aquí está el manifiesto ampliado con más principios y mejores prácticas de la industria para el desarrollo de microservicios con Java y Spring Boot:

Arquitectura y Tecnologías

1. Microservicios: La arquitectura seguirá el patrón de microservicios desarrollados en Java versión 21 con Spring Boot 3.x.
2. Contenedores Docker: Los microservicios se empaquetarán y desplegarán en contenedores Docker.

3. CI/CD: Se implementará un pipeline de integración y entrega continuas (CI/CD) con Jenkins.
4. Arquetipos Maven: Los nuevos microservicios se generarán a partir de arquetipos Maven.
5. Repositorio GitLab: El código fuente se gestionará en un repositorio GitLab.
6. Documentación OpenAPI: Las APIs se documentarán utilizando especificaciones OpenAPI.

Principios Generales

1. Dominio Acotado: Cada microservicio tendrá un dominio de negocio acotado y bien definido.
2. Responsabilidad Única: Los microservicios deben seguir el principio de responsabilidad única.
3. Desacoplamiento: Los microservicios estarán desacoplados y podrán ser desplegados de forma independiente.
4. Contratos Estrictos: Las interfaces entre microservicios tendrán contratos estrictos.
5. Redundancia Mínima: Se evitará la redundancia de código y datos entre microservicios.
6. Tolerancia a Fallos: Los microservicios serán resistentes a fallos y se recuperarán de forma elegante.

Arquitectura de Microservicios

1. Patrón API Gateway: Se utilizará un API Gateway para enrutar las solicitudes a los microservicios.
2. Service Discovery: Los microservicios utilizarán un mecanismo de descubrimiento de servicios.
3. Circuit Breaker: Se implementará un patrón de Circuit Breaker para evitar saturaciones.
4. Equilibrio de Carga: Se realizará un balanceo de carga entre las instancias de los microservicios.
5. Monitoreo Distribuido: Se monitoreará el estado y rendimiento de los microservicios de forma distribuida.
6. Registro Centralizado: Los registros (logs) de los microservicios se centralizarán para facilitar el análisis.

Diseño de Microservicios

1. Basado en Eventos: Los microservicios se comunicarán mediante un modelo basado en eventos cuando sea posible.
2. API RESTful: Las APIs de los microservicios seguirán los principios de diseño RESTful.
3. Versionamiento de APIs: Las APIs tendrán un esquema de versionamiento definido.
4. Stateless: Los microservicios serán stateless siempre que sea posible.
5. Almacenamiento Externo: Los microservicios no almacenarán datos localmente, utilizarán almacenes externos.
6. Escalabilidad: El diseño permitirá la escalabilidad horizontal de los microservicios según las necesidades.
7. Seguridad: Se implementarán mecanismos de seguridad como autenticación, autorización y cifrado de datos.
8. Diseño Guiado por Dominio: Aplicar los principios del Diseño Guiado por el Dominio (DDD).
9. Patrones de Diseño: Utilizar patrones de diseño apropiados como CQRS, Event Sourcing, Saga, etc.
10. Mensajería Asíncrona: Favorecer la comunicación asíncrona entre microservicios mediante mensajería.
11. Gestión de Errores: Implementar una gestión sólida de errores y excepciones.
12. Correlación de Trazas: Implementar un mecanismo para correlacionar trazas distribuidas.

13. Caching Distribuido: Utilizar una solución de caché distribuida para mejorar el rendimiento.
14. API Compuesta: Implementar una API compuesta cuando sea necesario.
15. Separación de Concerns: Separar claramente las responsabilidades de los microservicios.
16. Principio SOLID: Aplicar los principios SOLID en el diseño e implementación.

Desarrollo de Microservicios

1. Generación por Arquetipo: Todo microservicio del negocio se generará por medio de un arquetipo Maven.
2. Registro en Repositorio: Los microservicios se registrarán en el repositorio de la organización.
3. Perfiles de Configuración: Cada microservicio tendrá perfiles de configuración para los ambientes (dev, qa, pre, prod).
4. Perfiles CI/CD: Cada microservicio tendrá perfiles de configuración CI/CD para los ambientes.
5. Mapeo de Entidades: Las entidades de persistencia mapearán toda la clase que representan.
6. Consultas SQL Complejas: Se utilizarán entidades y anotaciones `@Immutable`, `@SubSelect` para consultas SQL complejas.
7. Uso de Entidades: Las entidades sólo se usarán en clases `@Component`, no en `@RestController` o `@Service`.
8. Responsabilidad de Persistencia: Cada entidad pertenecerá a un solo microservicio responsable de su persistencia.
9. Exposición con DTOs: Las entidades se expondrán mediante DTOs (Data Transfer Objects).
10. Código SQL Estándar: Las entidades utilizarán código SQL estándar, sin depender de un motor de base de datos específico.
11. Servicios Orquestadores: Las clases `@Service` solo orquestarán los métodos de la solución.
12. Controladores de API: Las clases `@RestController` solo expondrán endpoints de API.
13. Clonación de Objetos: No se utilizarán interfaces `Clonable`, se implementarán métodos de clonación.
14. Estándares de Codificación: Definir y adherirse a estándares de codificación consistentes.
15. Inyección de Dependencias: Utilizar inyección de dependencias para desacoplamiento y testabilidad.
16. Configuración Externalizada: Externalizar la configuración de los microservicios.
17. Gestión de Secretos: Implementar una solución segura para la gestión de secretos.
18. Registro Estructurado: Utilizar un formato de registro estructurado y estandarizado.
19. Correlación de Registros: Correlacionar los registros de diferentes microservicios.

Pruebas y Calidad

1. Pruebas Unitarias: Todos los microservicios tendrán un conjunto completo de pruebas unitarias automatizadas.
2. Pruebas de Integración: Se implementarán pruebas de integración para validar las interacciones entre microservicios.
3. Pruebas de Contrato: Se utilizarán pruebas de contrato para validar las interfaces entre microservicios.
4. Cobertura de Pruebas: Se establecerá un umbral mínimo de cobertura de pruebas para cada microservicio.
5. Análisis Estático de Código: Se realizará un análisis estático de código para detectar problemas y vulnerabilidades.
6. Revisiones de Código: Se implementarán revisiones de código por pares antes de integrar los

cambios.
7. Pruebas de Rendimiento: Se realizarán pruebas de rendimiento y carga para validar el comportamiento bajo estrés.
8. Pruebas de Seguridad: Se llevarán a cabo pruebas de seguridad para identificar y mitigar vulnerabilidades.
9. Control de Calidad: Se establecerá un proceso de control de calidad para validar los entregables.
10. Métricas de Calidad: Se definirán y monitorearán métricas de calidad para cada microservicio.
11. Pruebas de Regresión: Ejecutar pruebas de regresión automatizadas antes de despliegues.
12. Pruebas de Caja Negra: Implementar pruebas de caja negra para el comportamiento externo.
13. Pruebas de Estrés: Realizar pruebas de estrés y carga para validar condiciones extremas.
14. Pruebas de Caos: Considerar la implementación de pruebas de caos para validar resiliencia.
15. Gestión de Datos de Prueba: Implementar una estrategia para la gestión de datos de prueba.
16. Pruebas Exploratorias: Realizar pruebas exploratorias para identificar escenarios no previstos.
17. Pruebas de Usabilidad: Llevar a cabo pruebas de usabilidad para validar la experiencia de usuario.
18. Pruebas de Accesibilidad: Implementar pruebas de accesibilidad para garantizar la inclusión.

Operaciones y Despliegue

1. Contenedores Inmutables: Los contenedores Docker serán inmutables y se reconstruirán para cada despliegue.
2. Orquestación de Contenedores: Se utilizará una plataforma de orquestación de contenedores como Kubernetes.
3. Despliegues Automatizados: Los despliegues se realizarán de forma automatizada mediante el pipeline de CI/CD.
4. Entornos Separados: Existirán entornos separados para desarrollo, pruebas, pre-producción y producción.
5. Promoción de Entornos: Los microservicios se promoverán a través de los entornos de forma controlada.
6. Versionamiento de Artefactos: Los artefactos de los microservicios tendrán un esquema de versionamiento claro.
7. Registro de Artefactos: Los artefactos se almacenarán en un registro de artefactos centralizado.
8. Rollbacks: Se implementará un mecanismo para realizar rollbacks de forma sencilla en caso de problemas.
9. Actualizaciones Graduales: Se utilizarán estrategias de despliegue gradual para minimizar el impacto en producción.
10. Monitoreo de Recursos: Se monitoreará el uso de recursos (CPU, memoria, red, disco) de los microservicios.
11. Alertas y Notificaciones: Se configurarán alertas y notificaciones para detectar problemas de forma temprana.
12. Respaldo y Recuperación: Se implementarán mecanismos de respaldo y recuperación de datos.
13. Gestión de Incidentes: Establecer un proceso para la gestión y resolución de incidentes.
14. Gestión de Cambios: Implementar un proceso de gestión de cambios.
15. Documentación Actualizada: Mantener la documentación técnica y operativa actualizada.
16. Pruebas de Humo: Implementar pruebas de “humo” para validar los despliegues.
17. Canary Releases: Utilizar despliegues canarios para pruebas de producción controladas.
18. Separación de Responsabilidades: Separar responsabilidades de desarrollo, operaciones y seguridad.

Mejora Continua y Prácticas de Equipo

1. Capacitación Continua: Fomentar la capacitación continua del equipo.
2. Retrospectivas: Realizar retrospectivas periódicas para identificar áreas de mejora.
3. Gestión de Dependencias: Implementar un proceso para gestionar dependencias.
4. Automatización de Procesos: Automatizar los procesos de desarrollo, pruebas y despliegue.
5. Análisis de Impacto: Realizar un análisis de impacto antes de cambios significativos.

Recuerde: El Manifiesto de Codificación no es una lista rígida de reglas, sino una guía flexible que se adapta a las necesidades de cada proyecto. Adopte estos principios y adáptelos a su contexto para crear software de alta calidad que resista el paso del tiempo.

←Regresar