Manual del Curso

Manual del Estudiante

MODULO 1

TEMA 1.1: ¿QUÉ ES REALMENTE LA ARQUITECTURA?

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender ¿qué es realmente la arquitectura?. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

¿Qué es realmente la Arquitectura? es un concepto clave en arquitectura de software que...

Definición formal

> **¿Qué es realmente la Arquitectura?**: [Definición técnica precisa]

Propiedades fundamentales

1. **Estructura vs Comportamiento**: Descripción detallada
2. **Arquitectura como decisiones difíciles de cambiar**: Descripción detallada
3. **Documentación viva y ADRs**: Descripción detallada

Implementación práctica

Conceptos clave

Estructura vs Comportamiento

[Explicación detallada del subtema]

Arquitectura como decisiones difíciles de cambiar

[Explicación detallada del subtema]

Documentación viva y ADRs

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: ¿Qué es realmente la Arquitectura? es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ Estructura vs Comportamiento
2. ✅ Arquitectura como decisiones difíciles de cambiar
3. ✅ Documentación viva y ADRs

EJERCICIOS: TEMA 1.1 - ¿Qué es realmente la Arquitectura?

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es ¿Qué es realmente la Arquitectura? y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de ¿Qué es realmente la Arquitectura?
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar ¿Qué es realmente la Arquitectura? en un proyecto real

Requisitos:

1. Implementar Estructura vs Comportamiento
2. Implementar Arquitectura como decisiones difíciles de cambiar
3. Implementar Documentación viva y ADRs

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar ¿Qué es realmente la Arquitectura? para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 1.1 - ¿Qué es realmente la Arquitectura?

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre ¿Qué es realmente la Arquitectura? es correcta?

- [ ] **a)** Estructura vs Comportamiento
- [ ] **b)** Arquitectura como decisiones difíciles de cambiar
- [ ] **c)** Documentación viva y ADRs
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

¿Qué es realmente la Arquitectura? siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando ¿Qué es realmente la Arquitectura? en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice ¿Qué es realmente la Arquitectura? para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. Estructura vs Comportamiento
2. Arquitectura como decisiones difíciles de cambiar
3. Documentación viva y ADRs

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: ¿Qué es realmente la Arquitectura? abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: ¿Qué es realmente la Arquitectura? no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. Estructura vs Comportamiento
2. Arquitectura como decisiones difíciles de cambiar
3. Documentación viva y ADRs

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de ¿Qué es realmente la Arquitectura? (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

TEMA 1.1: ¿QUÉ ES REALMENTE LA ARQUITECTURA?

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Experiencia básica en desarrollo de software, conocimiento de patrones de diseño

¿Por qué importa este concepto?

La arquitectura de software es uno de los conceptos más malinterpretados en la industria. Muchos desarrolladores confunden arquitectura con diseño, o peor aún, con la elección de tecnologías. Esta confusión lleva a decisiones costosas que son difíciles de revertir.

Comprender qué es realmente la arquitectura te permite:

- Tomar decisiones que maximizan la flexibilidad futura del sistema
- Comunicar efectivamente las restricciones y trade-offs a stakeholders
- Distinguir entre decisiones arquitectónicas (difíciles de cambiar) y decisiones de diseño (fáciles de cambiar)

En sistemas reales, la arquitectura determina el 80% del costo total de mantenimiento. Un sistema con mala arquitectura puede funcionar perfectamente en producción, pero será imposible de evolucionar.

TEMA 1.2: ATRIBUTOS DE CALIDAD (-ILITIES)

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender atributos de calidad (-ilities). En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Atributos de Calidad (-ilities) es un concepto clave en arquitectura de software que...

Definición formal

> **Atributos de Calidad (-ilities)**: [Definición técnica precisa]

Propiedades fundamentales

1. **Scalability**: Descripción detallada
2. **Maintainability**: Descripción detallada
3. **Performance**: Descripción detallada
4. **Reliability**: Descripción detallada
5. **Security**: Descripción detallada
6. **Testability**: Descripción detallada
7. **Usability**: Descripción detallada

Implementación práctica

Conceptos clave

Scalability

[Explicación detallada del subtema]

Maintainability

[Explicación detallada del subtema]

Performance

[Explicación detallada del subtema]

Reliability

[Explicación detallada del subtema]

Security

[Explicación detallada del subtema]

Testability

[Explicación detallada del subtema]

Usability

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Atributos de Calidad (-ilities) es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ Scalability
2. ✅ Maintainability
3. ✅ Performance
4. ✅ Reliability
5. ✅ Security
6. ✅ Testability
7. ✅ Usability

EJERCICIOS: TEMA 1.2 - Atributos de Calidad (-ilities)

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Atributos de Calidad (-ilities) y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Atributos de Calidad (-ilities)
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Atributos de Calidad (-ilities) en un proyecto real

Requisitos:

1. Implementar Scalability
2. Implementar Maintainability
3. Implementar Performance
4. Implementar Reliability
5. Implementar Security
6. Implementar Testability
7. Implementar Usability

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Atributos de Calidad (-ilities) para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)

- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 1.2 - Atributos de Calidad (-ilities)

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Atributos de Calidad (-ilities) es correcta?

- [ ] **a)** Scalability
- [ ] **b)** Maintainability
- [ ] **c)** Performance
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Atributos de Calidad (-ilities) siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Atributos de Calidad (-ilities) en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Atributos de Calidad (-ilities) para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. Scalability
2. Maintainability
3. Performance
4. Reliability
5. Security
6. Testability
7. Usability

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Atributos de Calidad (-ilities) abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Atributos de Calidad (-ilities) no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. Scalability
2. Maintainability
3. Performance

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Atributos de Calidad (-ilities) (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

TEMA 1.3: LEY DE CONWAY Y DISEÑO ORGANIZACIONAL

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender ley de conway y diseño organizacional. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Ley de Conway y Diseño Organizacional es un concepto clave en arquitectura de software que...

Definición formal

> **Ley de Conway y Diseño Organizacional**: [Definición técnica precisa]

Propiedades fundamentales

1. **Implicaciones en microservicios**: Descripción detallada
2. **Inversión del Diseño de Conway**: Descripción detallada

Implementación práctica

Conceptos clave

Implicaciones en microservicios

[Explicación detallada del subtema]

Inversión del Diseño de Conway

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Ley de Conway y Diseño Organizacional es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ Implicaciones en microservicios
2. ✅ Inversión del Diseño de Conway

EJERCICIOS: TEMA 1.3 - Ley de Conway y Diseño Organizacional

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Ley de Conway y Diseño Organizacional y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Ley de Conway y Diseño Organizacional
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Ley de Conway y Diseño Organizacional en un proyecto real

Requisitos:

1. Implementar Implicaciones en microservicios
2. Implementar Inversión del Diseño de Conway

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Ley de Conway y Diseño Organizacional para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 1.3 - Ley de Conway y Diseño Organizacional

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Ley de Conway y Diseño Organizacional es correcta?

- [ ] **a)** Implicaciones en microservicios
- [ ] **b)** Inversión del Diseño de Conway
- [ ] **c)** Opción C
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Ley de Conway y Diseño Organizacional siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Ley de Conway y Diseño Organizacional en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Ley de Conway y Diseño Organizacional para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. Implicaciones en microservicios
2. Inversión del Diseño de Conway

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Ley de Conway y Diseño Organizacional abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Ley de Conway y Diseño Organizacional no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. Implicaciones en microservicios
2. Inversión del Diseño de Conway
3. Aspecto 3

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Ley de Conway y Diseño Organizacional (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

TEMA 1.4: ARQUITECTO COMO LÍDER TÉCNICO

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender arquitecto como líder técnico. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Arquitecto como Líder Técnico es un concepto clave en arquitectura de software que...

Definición formal

> **Arquitecto como Líder Técnico**: [Definición técnica precisa]

Propiedades fundamentales

1. **Comunicación**: Descripción detallada
2. **Mentoría**: Descripción detallada
3. **Toma de Decisiones**: Descripción detallada

Implementación práctica

Conceptos clave

Comunicación

[Explicación detallada del subtema]

Mentoría

[Explicación detallada del subtema]

Toma de Decisiones

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Arquitecto como Líder Técnico es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ Comunicación
2. ✅ Mentoría
3. ✅ Toma de Decisiones

EJERCICIOS: TEMA 1.4 - Arquitecto como Líder Técnico

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Arquitecto como Líder Técnico y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso

- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Arquitecto como Líder Técnico
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Arquitecto como Líder Técnico en un proyecto real

Requisitos:

1. Implementar Comunicación
2. Implementar Mentoría
3. Implementar Toma de Decisiones

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Arquitecto como Líder Técnico para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 1.4 - Arquitecto como Líder Técnico

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Arquitecto como Líder Técnico es correcta?

- [ ] **a)** Comunicación
- [ ] **b)** Mentoría
- [ ] **c)** Toma de Decisiones
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Arquitecto como Líder Técnico siempre mejora el rendimiento del sistema.

- [ ] Verdadero

- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Arquitecto como Líder Técnico en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Arquitecto como Líder Técnico para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. Comunicación
2. Mentoría
3. Toma de Decisiones

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Respuesta 2

Correcta: Falso

Justificación: Arquitecto como Líder Técnico no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. Comunicación
2. Mentoría
3. Toma de Decisiones

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Arquitecto como Líder Técnico (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

MODULO 2

TEMA 2.1: MODULARIDAD EFICAZ

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender modularidad eficaz. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Modularidad Eficaz es un concepto clave en arquitectura de software que...

Definición formal

> **Modularidad Eficaz**: [Definición técnica precisa]

Propiedades fundamentales

1. **Acoplamiento**: Descripción detallada
2. **Cohesión**: Descripción detallada
3. **Separación de Responsabilidades**: Descripción detallada
4. **Bounded Context**: Descripción detallada

Implementación práctica

Conceptos clave

Acoplamiento

[Explicación detallada del subtema]

Cohesión

[Explicación detallada del subtema]

Separación de Responsabilidades

[Explicación detallada del subtema]

Bounded Context

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Modularidad Eficaz es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ Acoplamiento
2. ✅ Cohesión
3. ✅ Separación de Responsabilidades
4. ✅ Bounded Context

EJERCICIOS: TEMA 2.1 - Modularidad Eficaz

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Modularidad Eficaz y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Modularidad Eficaz
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Modularidad Eficaz en un proyecto real

Requisitos:

1. Implementar Acoplamiento
2. Implementar Cohesión
3. Implementar Separación de Responsabilidades
4. Implementar Bounded Context

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Modularidad Eficaz para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 2.1 - Modularidad Eficaz

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse

- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Modularidad Eficaz es correcta?

- [ ] **a)** Acoplamiento
- [ ] **b)** Cohesión
- [ ] **c)** Separación de Responsabilidades
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Modularidad Eficaz siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Modularidad Eficaz en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Modularidad Eficaz para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. Acoplamiento
2. Cohesión
3. Separación de Responsabilidades
4. Bounded Context

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Modularidad Eficaz abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Modularidad Eficaz no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. Acoplamiento
2. Cohesión
3. Separación de Responsabilidades

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Modularidad Eficaz (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

TEMA 2.2: PRINCIPIOS SOLID EN ARQUITECTURA

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender principios solid en arquitectura. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Principios SOLID en Arquitectura es un concepto clave en arquitectura de software que...

Definición formal

> **Principios SOLID en Arquitectura**: [Definición técnica precisa]

Propiedades fundamentales

1. **Interpretación práctica a nivel de componentes y módulos**: Descripción detallada

Implementación práctica

Interpretación práctica a nivel de componentes y módulos

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Principios SOLID en Arquitectura es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ Interpretación práctica a nivel de componentes y módulos

EJERCICIOS: TEMA 2.2 - Principios SOLID en Arquitectura

Ejercicios Conceptuales

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Principios SOLID en Arquitectura y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Principios SOLID en Arquitectura
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Principios SOLID en Arquitectura en un proyecto real

Requisitos:

1. Implementar Interpretación práctica a nivel de componentes y módulos

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Principios SOLID en Arquitectura para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 2.2 - Principios SOLID en Arquitectura

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Principios SOLID en Arquitectura es correcta?

- [ ] **a)** Interpretación práctica a nivel de componentes y módulos
- [ ] **b)** Opción B
- [ ] **c)** Opción C
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Principios SOLID en Arquitectura siempre mejora el rendimiento del sistema.

- [ ] Verdadero

- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Principios SOLID en Arquitectura en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Principios SOLID en Arquitectura para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. Interpretación práctica a nivel de componentes y módulos

SOLUCIONARIO

Correcta: d) Todas las anteriores

Explicación: Principios SOLID en Arquitectura abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Principios SOLID en Arquitectura no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. Interpretación práctica a nivel de componentes y módulos
2. Aspecto 2
3. Aspecto 3

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Principios SOLID en Arquitectura (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

TEMA 2.3: ARQUITECTURA LIMPIA Y HEXAGONAL

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender arquitectura limpia y hexagonal. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Arquitectura Limpia y Hexagonal es un concepto clave en arquitectura de software que...

Definición formal

> **Arquitectura Limpia y Hexagonal**: [Definición técnica precisa]

Propiedades fundamentales

1. **Ports & Adapters**: Descripción detallada
2. **Dependencia hacia adentro**: Descripción detallada
3. **Testing con dobles**: Descripción detallada

Implementación práctica

Conceptos clave

Ports & Adapters

[Explicación detallada del subtema]

Dependencia hacia adentro

[Explicación detallada del subtema]

Testing con dobles

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Arquitectura Limpia y Hexagonal es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ Ports & Adapters
2. ✅ Dependencia hacia adentro
3. ✅ Testing con dobles

EJERCICIOS: TEMA 2.3 - Arquitectura Limpia y Hexagonal

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Arquitectura Limpia y Hexagonal y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Arquitectura Limpia y Hexagonal
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Arquitectura Limpia y Hexagonal en un proyecto real

Requisitos:

1. Implementar Ports & Adapters
2. Implementar Dependencia hacia adentro
3. Implementar Testing con dobles

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Arquitectura Limpia y Hexagonal para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 2.3 - Arquitectura Limpia y Hexagonal

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Arquitectura Limpia y Hexagonal es correcta?

- [ ] **a)** Ports & Adapters
- [ ] **b)** Dependencia hacia adentro
- [ ] **c)** Testing con dobles
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Arquitectura Limpia y Hexagonal siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Arquitectura Limpia y Hexagonal en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Arquitectura Limpia y Hexagonal para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. Ports & Adapters
2. Dependencia hacia adentro
3. Testing con dobles

SOLUCIONARIO

Respuesta 1

Explicación: Arquitectura Limpia y Hexagonal abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Arquitectura Limpia y Hexagonal no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. Ports & Adapters
2. Dependencia hacia adentro
3. Testing con dobles

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Arquitectura Limpia y Hexagonal (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

TEMA 2.4: DEUDA TÉCNICA

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender deuda técnica. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Deuda Técnica es un concepto clave en arquitectura de software que...

Definición formal

> **Deuda Técnica**: [Definición técnica precisa]

Propiedades fundamentales

1. **Tipos de Deuda**: Descripción detallada
2. **Priorización y Pago**: Descripción detallada
3. **Deuda Arquitectónica**: Descripción detallada

Implementación práctica

Conceptos clave

Tipos de Deuda

[Explicación detallada del subtema]

Priorización y Pago

[Explicación detallada del subtema]

Deuda Arquitectónica

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Deuda Técnica es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ Tipos de Deuda
2. ✅ Priorización y Pago
3. ✅ Deuda Arquitectónica

EJERCICIOS: TEMA 2.4 - Deuda Técnica

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Deuda Técnica y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Deuda Técnica
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Deuda Técnica en un proyecto real

Requisitos:

1. Implementar Tipos de Deuda
2. Implementar Priorización y Pago
3. Implementar Deuda Arquitectónica

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Deuda Técnica para resolver el problema

Rúbrica:

- Análisis del problema (25%)

- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 2.4 - Deuda Técnica

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Deuda Técnica es correcta?

- [ ] **a)** Tipos de Deuda
- [ ] **b)** Priorización y Pago
- [ ] **c)** Deuda Arquitectónica
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Deuda Técnica siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Deuda Técnica en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Deuda Técnica para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. Tipos de Deuda
2. Priorización y Pago
3. Deuda Arquitectónica

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Deuda Técnica abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Deuda Técnica no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. Tipos de Deuda
2. Priorización y Pago
3. Deuda Arquitectónica

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Deuda Técnica (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

MODULO 3

TEMA 3.1: LAYERED ARCHITECTURE (CAPAS)

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender layered architecture (capas). En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Layered Architecture (Capas) es un concepto clave en arquitectura de software que...

Definición formal

> **Layered Architecture (Capas)**: [Definición técnica precisa]

Propiedades fundamentales

1. **El estándar de facto**: Descripción detallada
2. **Capa de Dominio**: Descripción detallada
3. **Capa de Aplicación**: Descripción detallada
4. **Capa de Infraestructura**: Descripción detallada

Implementación práctica

Conceptos clave

El estándar de facto

[Explicación detallada del subtema]

Capa de Dominio

[Explicación detallada del subtema]

Capa de Aplicación

[Explicación detallada del subtema]

Capa de Infraestructura

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Layered Architecture (Capas) es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

1. ✅ El estándar de facto
2. ✅ Capa de Dominio
3. ✅ Capa de Aplicación
4. ✅ Capa de Infraestructura

EJERCICIOS: TEMA 3.1 - Layered Architecture (Capas)

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Layered Architecture (Capas) y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Layered Architecture (Capas)
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Layered Architecture (Capas) en un proyecto real

Requisitos:

1. Implementar El estándar de facto
2. Implementar Capa de Dominio
3. Implementar Capa de Aplicación
4. Implementar Capa de Infraestructura

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Layered Architecture (Capas) para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 3.1 - Layered Architecture (Capas)

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Layered Architecture (Capas) es correcta?

- [ ] **a)** El estándar de facto
- [ ] **b)** Capa de Dominio
- [ ] **c)** Capa de Aplicación
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Layered Architecture (Capas) siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Layered Architecture (Capas) en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Layered Architecture (Capas) para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. El estándar de facto
2. Capa de Dominio
3. Capa de Aplicación
4. Capa de Infraestructura

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Layered Architecture (Capas) abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Layered Architecture (Capas) no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. El estándar de facto
2. Capa de Dominio
3. Capa de Aplicación

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Layered Architecture (Capas) (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

TEMA 3.1: LAYERED ARCHITECTURE (CAPAS)

Tiempo estimado: 45 minutos **Nivel**: Intermedio **Prerrequisitos**: Módulo 2

1. El Estándar de Facto

Es la arquitectura más común. Todo el mundo la conoce. Se organiza en capas horizontales, donde cada capa tiene un rol específico.

2. Las Capas Clásicas

1. **Presentation (UI)**: Maneja HTML, JSON, CLI. No tiene lógica de negocio.
2. **Business Logic (Domain)**: Toma decisiones. Calcula impuestos, valida reglas.
3. **Persistence (Data Access)**: SQL, ORMs. Sabe cómo guardar cosas.
4. **Database**: El almacenamiento físico.

3. Reglas de Oro

- **Separation of Concerns**: La UI no debe hacer consultas SQL. La BD no debe renderizar HTML.
- **Layers of Isolation**: Una capa solo puede llamar a la capa inmediatamente inferior (Strict) o a cualquiera inferior (Relaxed).

4. Anti-Patrón: Architecture Sinkhole

Cuando las peticiones pasan por las capas sin hacer nada. UI → Business (Pass-through) → Persistence (Pass-through) → DB. El 80% de tu código son funciones que solo llaman a otra función. Si tienes mucho de esto, quizás Layered no es para ti.

Resumen

Ideal para equipos nuevos, aplicaciones simples y prototipos. Malo para despliegues rápidos (hay que compilar todo el monolito) y escalabilidad granular.

EJERCICIOS: TEMA 3.1 (LAYERED ARCHITECTURE)

OBJETIVOS

1. Identificar violaciones de capas.
2. Refactorizar código espagueti a capas.

---

1. POLICÍA DE CAPAS

Código Sospechoso (Controller.js):

```
router.post('/checkout', (req, res) => {
    // 1. Validar input (OK para Controller)
    if (!req.body.cart) return res.status(400);

    // 2. Calcular total con impuestos (BLOQUEO: Lógica de Negocio)
    let total = 0;
    req.body.cart.forEach(item => {
        if (item.category === 'food') total += item.price * 1.10;
        else total += item.price * 1.21;
    });

db.query("INSERT INTO orders...", [total]);

    res.send({ total });
});
```

Tarea: Separa este código en 3 archivos ficticios: `OrderController`, `OrderService`, `OrderRepository`.

---

2. STRICT VS RELAXED

Si tienes una capa de "Utils" (ej: formatear fechas), ¿quién puede llamarla?

- ¿Solo Persistence?
- ¿Solo Business?
- ¿Todos? Dibuja un diagrama de dependencias si permitimos que TODOS llamen a Utils (Relaxed Layering). ¿Crea esto acoplamiento peligroso?

EVALUACIÓN: TEMA 3.1 (LAYERED ARCHITECTURE)

FICHA TÉCNICA

- **Tema**: Patrones Monolíticos
- **Nivel**: Intermedio

CUESTIONARIO

Pregunta 1

¿Cuál es la responsabilidad principal de la capa de **Lógica de Negocio** (Domain)?

- [ ] **a)** Mostrar los datos al usuario en HTML.
- [ ] **b)** Ejecutar reglas empresariales, cálculos y validaciones (ej: calcular impuestos).
- [ ] **c)** Ejecutar consultas SQL directas contra la base de datos.

¿Qué problema describe el anti-patrón “Architecture Sinkhole”?

- [ ] **a)** Cuando la base de datos se llena.
- [ ] **b)** Cuando las peticiones atraviesan múltiples capas sin que ninguna aporte valor añadido (solo pass-through code).
- [ ] **c)** Cuando hay agujeros de seguridad.

Pregunta 3

En un modelo de “Capas Estrictas” (Strict Layering), la capa de Presentación puede llamar a:

- [ ] **a)** Solo a la capa de Negocio (la inmediatamente inferior).
- [ ] **b)** A la capa de Negocio y a la de Persistencia.
- [ ] **c)** Directamente a la Base de Datos.

Pregunta 4

Si cambias tu motor de base de datos de Oracle a PostgreSQL, ¿qué capas deberían verse afectadas idealmente?

- [ ] **a)** Todas (Presentación, Negocio y Persistencia).
- [ ] **b)** Solo la capa de Persistencia (y Configuración).
- [ ] **c)** Solo la capa de Presentación.

SOLUCIONARIO

1. **b)**. Es el cerebro de la aplicación.
2. **b)**. Overhead innecesario.
3. **a)**. Mantiene el desacoplamiento máximo.
4. **b)**. Gracias al aislamiento de capas.

TEMA 3.2: MICROKERNEL (PLUG-IN ARCHITECTURE)

Tiempo estimado: 45 minutos **Nivel**: Avanzado **Prerrequisitos**: Tema 3.1

1. El Problema de la Personalización

Tienes una app que usan 100 clientes distintos. El Cliente A quiere pagar con Paypal. El B con Stripe. El C con Bitcoin. Si pones TODA esa lógica en el código principal (`if (client == A) ...`), creas un monstruo inmanejable.

2. La Solución: Microkernel

Divides el sistema en dos partes:

1. **Core System (Microkernel)**: La lógica mínima indispensable que comparten todos. (Ej: "Procesar Pedido").
2. **Plug-ins Modules**: Componentes independientes que añaden funcionalidad específica. (Ej: "Plugin Paypal", "Plugin Stripe").

3. Ejemplo: VS Code

VS Code es un editor de texto simple (Core). Todo lo demás (Python support, GitLens, Themes) son extensiones (Plugins). El Core no sabe nada de Python. Solo sabe "Cargar extensión" y "Ejecutar comando".

4. Contratos (API)

Para que funcione, el Core debe definir un Contrato estricto (Interfaz). `interface IPaymentPlugin { function pay(amount); }` Cualquier plugin que respete esa firma puede enchufarse.

Resumen

Útil para software de escritorio, navegadores y productos SaaS muy customizables. Difícil de implementar porque el Core debe ser muy estable (si cambia el API del Core, rompes 1000 plugins).

EJERCICIOS: TEMA 3.2 (MICROKERNEL)

OBJETIVOS

1. Diseñar una arquitectura extensible.
2. Definir contratos de Plugin.

---

1. EL NAVEGADOR WEB

Estás construyendo un navegador nuevo ("ChromeKiller"). El navegador base solo sabe renderizar HTML. Quieres permitir que terceros creen "AdBlockers" o "DarkReaders".

Tarea: Diseña la interfaz del Plugin.

```
interface IBrowserPlugin {
    // ¿Qué métodos necesita exponer el plugin?
    onPageLoad(url: string, content: string): string; // Para modificar contenido
    onRequest(url: string): boolean; // Para bloquear traza
}
```

---

2. EL REGISTRY

¿Cómo sabe el Core qué plugins están instalados? Escribe pseudocódigo para un `PluginRegistry`.

```
class Registry {
    plugins = [];

    register(plugin) {
        // Validar que cumpla la interfaz
        ____________________
```

```
        this.plugins.push(plugin);
    }

    executeAll(eventName, data) {
        // Recorrer plugins y ejecutarlos
        ____________________
    }
}
```

---

3. REFLEXIÓN

Si un Plugin falla (lanza excepción infinita), ¿debe caerse el Core? ¿Cómo aíslas el Plugin? (Ej: Ejecutar en un Sandbox, WebWorker o proceso separado).

EVALUACIÓN: TEMA 3.2 (MICROKERNEL)

FICHA TÉCNICA

- **Tema**: Arquitectura de Plugins
- **Nivel**: Avanzado

CUESTIONARIO

Pregunta 1

En una arquitectura Microkernel, ¿cuál es la relación de dependencia correcta?

- [ ] **a)** El Core depende de los Plugins (el Core importa los plugins).
- [ ] **b)** Los Plugins dependen del Core (los Plugins importan la API del Core).
- [ ] **c)** Son interdependientes (Circular).

Pregunta 2

¿Cuál es la función del “Plugin Registry”?

- [ ] **a)** Cobrar dinero por los plugins.
- [ ] **b)** Mantener un inventario de qué plugins están disponibles y cómo cargarlos/instanciarlos.

- [ ] **c)** Validar el código fuente.

Pregunta 3

Si modificas la API pública del Core (Breaking Change), ¿qué consecuencia tiene?

- [ ] **a)** Ninguna, es transparente.
- [ ] **b)** Probablemente rompas todos los plugins existentes que usaban esa API, obligando a sus autores a actualizarlos.
- [ ] **c)** El sistema se vuelve más rápido.

Pregunta 4

El "Core System" debería contener:

- [ ] **a)** Toda la lógica de negocio posible para cubrir todos los casos extremos.
- [ ] **b)** Solo la lógica mínima indispensable y común para que el sistema arranque y pueda ejecutar las extensiones.
- [ ] **c)** Solo la base de datos.

SOLUCIONARIO

1. **b)**. El Core debe ser agnóstico.
2. **b)**. Es el portero de la discoteca.
3. **b)**. Por eso el Core debe ser muy estable.
4. **b)**. "Keep it small".

TEMA 3.3: PIPELINE (PIPES AND FILTERS)

Tiempo estimado: 45 minutos **Nivel**: Intermedio **Prerrequisitos**: Tema 3.1

1. La Tubería de Datos

A veces tu aplicación no es interactiva. Es solo "procesar datos". Entra un archivo CSV → Se valida → Se transforma → Se guarda en BD. Esto es un Pipeline.

2. Componentes

1. **Filter (Filtro)**: Un componente independiente que hace UNA sola cosa.
    - `Filter A`: Convierte mayúsculas.
    - `Filter B`: Elimina duplicados.
2. **Pipe (Tubería)**: El conector que lleva el output de A al input de B.

3. Ejemplo: Unix

El mejor ejemplo del mundo. `cat archivo.txt | grep "error" | sort | uniq > reporte.txt` Cada comando es un filtro. | es el pipe. Los filtros no se conocen entre sí. Solo esperan texto en STDIN y escupen texto en STDOUT.

4. Pros y Contras

- **Pros**: Reutilización extrema. Puedes recombinar filtros en cualquier orden. Paralelismo (cada filtro puede correr en su propio hilo).
- **Contras**: Overhead de parseo (A escupe JSON, B tiene que leer JSON). Manejo de errores global complicado.

Resumen

Ideal para ETLs (Extract, Transform, Load), compiladores y procesamiento de video. No lo uses para una web interactiva compleja.

EJERCICIOS: TEMA 3.3 (PIPELINE)

OBJETIVOS

1. Construir una cadena de procesamiento.
2. Entender la independencia de los filtros.

---

1. COMPOSICIÓN FUNCIONAL

Tarea: Implementa un pipeline simple en JavaScript usando composición de funciones.

```
const text = "   Hola Mundo  ";
```

```
const trim = (str) => str.trim();
const toLower = (str) => str.toLowerCase();
const wrapP = (str) => `<p>${str}</p>`;

// Tu tarea: Crea una función `pipeline` que tome N filtros y los ejecute en orden.
const process = pipeline(trim, toLower, wrapP);

console.log(process(text)); // "<p>hola mundo</p>"
```

---

2. STREAMING

un script teórico que lea `bigdata.csv`, comprima con Gzip y escriba en disco, todo streamado.

```
fs.createReadStream('input.csv')
  .pipe(____________________) // Comprimir
  .pipe(fs.createWriteStream('output.gz'));
```

---

3. REFLEXIÓN

Si el filtro 3 falla (ej: formato inválido), ¿cómo le avisa al filtro 1 para que deje de enviar datos?

EVALUACIÓN: TEMA 3.3 (PIPELINE)

FICHA TÉCNICA

- **Tema**: Procesamiento de Datos
- **Nivel**: Intermedio

CUESTIONARIO

Pregunta 1

¿Qué caracteriza a un “Filtro” en este patrón?

- [ ] **a)** Es un componente que conoce a todos los demás componentes del sistema.
- [ ] **b)** Es un componente independiente que transforma datos de entrada en datos de salida sin conocer la identidad de sus vecinos.
- [ ] **c)** Es un componente de base de datos.

Pregunta 2

¿Cuál es una ventaja clave del patrón Pipeline?

- [ ] **a)** Interfaz de usuario rica.
- [ ] **b)** Reutilización y recomposición (puedes reorganizar los filtros para crear nuevos flujos).
- [ ] **c)** Baja latencia para peticiones web síncronas.

Pregunta 3

¿Qué analogía describe mejor este patrón?

- [ ] **a)** Una partida de ajedrez.
- [ ] **b)** Una línea de montaje industrial (fábrica).
- [ ] **c)** Una biblioteca.

Pregunta 4

¿Para qué tipo de problema es ideal este patrón?

- [ ] **a)** Aplicaciones CRUD simples.
- [ ] **b)** Procesamiento ETL (Extract, Transform, Load), compiladores, y procesamiento multimedia.
- [ ] **c)** Juegos en tiempo real.

SOLUCIONARIO

1. **b)**. Ignorancia del contexto.
2. **b)**. Como piezas de Lego.
3. **b)**. Fordismo aplicado al software.
4. **b)**. Flujos de transformación lineal.

TEMA 3.4: MONOLITO MODULAR

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender monolito modular. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Monolito Modular es un concepto clave en arquitectura de software que...

Definición formal

> **Monolito Modular**: [Definición técnica precisa]

Propiedades fundamentales

1. **Separación estricta de módulos**: Descripción detallada
2. **Comunicación interna organizada**: Descripción detallada

Implementación práctica

Conceptos clave

Separación estricta de módulos

[Explicación detallada del subtema]

Comunicación interna organizada

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Monolito Modular es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ Separación estricta de módulos
2. ✅ Comunicación interna organizada

EJERCICIOS: TEMA 3.4 - Monolito Modular

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Monolito Modular y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Monolito Modular
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Monolito Modular en un proyecto real

Requisitos:

1. Implementar Separación estricta de módulos
2. Implementar Comunicación interna organizada

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Monolito Modular para resolver el problema

Rúbrica:

- Análisis del problema (25%)

- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 3.4 - Monolito Modular

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Monolito Modular es correcta?

- [ ] **a)** Separación estricta de módulos
- [ ] **b)** Comunicación interna organizada
- [ ] **c)** Opción C
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Monolito Modular siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Monolito Modular en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Monolito Modular para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. Separación estricta de módulos
2. Comunicación interna organizada

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Monolito Modular abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Monolito Modular no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. Separación estricta de módulos
2. Comunicación interna organizada
3. Aspecto 3

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Monolito Modular (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

MODULO 4

TEMA 4.1: FALACIAS DE LA COMPUTACIÓN DISTRIBUIDA

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender falacias de la computación distribuida. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Falacias de la Computación Distribuida es un concepto clave en arquitectura de software que...

Definición formal

> **Falacias de la Computación Distribuida**: [Definición técnica precisa]

Propiedades fundamentales

1. **Latencia no es cero**: Descripción detallada
2. **La red no es confiable**: Descripción detallada
3. **Topologías cambiantes**: Descripción detallada

Implementación práctica

Conceptos clave

Latencia no es cero

[Explicación detallada del subtema]

La red no es confiable

[Explicación detallada del subtema]

Topologías cambiantes

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Falacias de la Computación Distribuida es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ Latencia no es cero
2. ✅ La red no es confiable

3. ✅ Topologías cambiantes

EJERCICIOS: TEMA 4.1 - Falacias de la Computación Distribuida

---

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Falacias de la Computación Distribuida y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Falacias de la Computación Distribuida
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Falacias de la Computación Distribuida en un proyecto real

Requisitos:

1. Implementar Latencia no es cero
2. Implementar La red no es confiable
3. Implementar Topologías cambiantes

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Falacias de la Computación Distribuida para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 4.1 - Falacias de la Computación Distribuida

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Falacias de la Computación Distribuida es correcta?

- [ ] **a)** Latencia no es cero
- [ ] **b)** La red no es confiable
- [ ] **c)** Topologías cambiantes
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Falacias de la Computación Distribuida siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Falacias de la Computación Distribuida en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Falacias de la Computación Distribuida para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. Latencia no es cero
2. La red no es confiable
3. Topologías cambiantes

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Falacias de la Computación Distribuida abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Falacias de la Computación Distribuida no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. Latencia no es cero
2. La red no es confiable
3. Topologías cambiantes

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Falacias de la Computación Distribuida (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

TEMA 4.2: MICROSERVICIOS

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender microservicios. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Microservicios es un concepto clave en arquitectura de software que...

Definición formal

Microservicios: [Definición técnica precisa]

Propiedades fundamentales

1. **Definición**: Descripción detallada
2. **Peligros**: Descripción detallada
3. **Diseño en torno a Bounded Contexts**: Descripción detallada
4. **Anti-patrones de microservicios**: Descripción detallada

Conceptos clave

Definición

[Explicación detallada del subtema]

Peligros

[Explicación detallada del subtema]

Diseño en torno a Bounded Contexts

[Explicación detallada del subtema]

Anti-patrones de microservicios

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Microservicios es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ Definición
2. ✅ Peligros
3. ✅ Diseño en torno a Bounded Contexts
4. ✅ Anti-patrones de microservicios

EJERCICIOS: TEMA 4.2 - Microservicios

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Microservicios y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Microservicios
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Microservicios en un proyecto real

Requisitos:

1. Implementar Definición
2. Implementar Peligros
3. Implementar Diseño en torno a Bounded Contexts
4. Implementar Anti-patrones de microservicios

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Microservicios para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 4.2 - Microservicios

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Microservicios es correcta?

- [ ] **a)** Definición
- [ ] **b)** Peligros
- [ ] **c)** Diseño en torno a Bounded Contexts
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Microservicios siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Microservicios en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Microservicios para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. Definición
2. Peligros
3. Diseño en torno a Bounded Contexts
4. Anti-patrones de microservicios

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Microservicios abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Microservicios no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. Definición
2. Peligros

3. Diseño en torno a Bounded Contexts

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Microservicios (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

TEMA 4.3: SERVICE-BASED ARCHITECTURE

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender service-based architecture. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Service-Based Architecture es un concepto clave en arquitectura de software que...

Definición formal

> **Service-Based Architecture**: [Definición técnica precisa]

Propiedades fundamentales

1. **El punto medio**: Descripción detallada
2. **Servicios coarse-grained**: Descripción detallada
3. **Menos overhead que microservicios**: Descripción detallada

Implementación práctica

Conceptos clave

El punto medio

[Explicación detallada del subtema]

Servicios coarse-grained

[Explicación detallada del subtema]

Menos overhead que microservicios

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Service-Based Architecture es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ El punto medio
2. ✅ Servicios coarse-grained

3. ✅ Menos overhead que microservicios

EJERCICIOS: TEMA 4.3 - Service-Based Architecture

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Service-Based Architecture y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Service-Based Architecture
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Requisitos:

1. Implementar El punto medio
2. Implementar Servicios coarse-grained
3. Implementar Menos overhead que microservicios

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Service-Based Architecture para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 4.3 - Service-Based Architecture

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Service-Based Architecture es correcta?

- [ ] **a)** El punto medio
- [ ] **b)** Servicios coarse-grained
- [ ] **c)** Menos overhead que microservicios
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Service-Based Architecture siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Service-Based Architecture en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Service-Based Architecture para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. El punto medio
2. Servicios coarse-grained
3. Menos overhead que microservicios

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Service-Based Architecture abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Service-Based Architecture no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. El punto medio
2. Servicios coarse-grained
3. Menos overhead que microservicios

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Service-Based Architecture (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

TEMA 4.4: MONOLITO DISTRIBUIDO (ANTI-PATRÓN)

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender monolito distribuido (anti-patrón). En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Monolito Distribuido (anti-patrón) es un concepto clave en arquitectura de software que...

Definición formal

> **Monolito Distribuido (anti-patrón)**: [Definición técnica precisa]

Propiedades fundamentales

1. **Síntomas**: Descripción detallada
2. **Cómo evitarlo**: Descripción detallada

Implementación práctica

Conceptos clave

Síntomas

[Explicación detallada del subtema]

Cómo evitarlo

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Monolito Distribuido (anti-patrón) es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ Síntomas
2. ✅ Cómo evitarlo

EJERCICIOS: TEMA 4.4 - Monolito Distribuido (anti-patrón)

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Monolito Distribuido (anti-patrón) y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Monolito Distribuido (anti-patrón)
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Monolito Distribuido (anti-patrón) en un proyecto real

Requisitos:

1. Implementar Síntomas
2. Implementar Cómo evitarlo

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Monolito Distribuido (anti-patrón) para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 4.4 - Monolito Distribuido (anti-patrón)

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Monolito Distribuido (anti-patrón) es correcta?

- [ ] **a)** Síntomas
- [ ] **b)** Cómo evitarlo
- [ ] **c)** Opción C
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Monolito Distribuido (anti-patrón) siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Monolito Distribuido (anti-patrón) en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Monolito Distribuido (anti-patrón) para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. Síntomas
2. Cómo evitarlo

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Monolito Distribuido (anti-patrón) abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Monolito Distribuido (anti-patrón) no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. Síntomas
2. Cómo evitarlo
3. Aspecto 3

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Monolito Distribuido (anti-patrón) (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

MODULO 5

TEMA 5.1: SÍNCRONO VS ASÍNCRONO

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender síncrono vs asíncrono. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Síncrono vs Asíncrono es un concepto clave en arquitectura de software que...

Definición formal

> **Síncrono vs Asíncrono**: [Definición técnica precisa]

Propiedades fundamentales

1. **REST**: Descripción detallada
2. **gRPC**: Descripción detallada
3. **WebSockets**: Descripción detallada
4. **Mensajería asíncrona**: Descripción detallada

Implementación práctica

Conceptos clave

REST

[Explicación detallada del subtema]

gRPC

[Explicación detallada del subtema]

WebSockets

[Explicación detallada del subtema]

Mensajería asíncrona

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Síncrono vs Asíncrono es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ REST
2. ✅ gRPC

3. ✅ WebSockets
4. ✅ Mensajería asíncrona

EJERCICIOS: TEMA 5.1 - Síncrono vs Asíncrono

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Síncrono vs Asíncrono y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Síncrono vs Asíncrono
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Síncrono vs Asíncrono en un proyecto real

Requisitos:

1. Implementar REST
2. Implementar gRPC
3. Implementar WebSockets
4. Implementar Mensajería asíncrona

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Síncrono vs Asíncrono para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 5.1 - Síncrono vs Asíncrono

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Síncrono vs Asíncrono es correcta?

- [ ] **a)** REST
- [ ] **b)** gRPC
- [ ] **c)** WebSockets
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Síncrono vs Asíncrono siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Síncrono vs Asíncrono en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Síncrono vs Asíncrono para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. REST
2. gRPC
3. WebSockets
4. Mensajería asíncrona

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Síncrono vs Asíncrono abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Síncrono vs Asíncrono no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. REST
2. gRPC
3. WebSockets

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Síncrono vs Asíncrono (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

TEMA 5.2: EVENT-DRIVEN ARCHITECTURE (EDA) Y EVENT SOURCING

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender event-driven architecture (eda) y event sourcing. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Event-Driven Architecture (EDA) y Event Sourcing es un concepto clave en arquitectura de software que...

Definición formal

> **Event-Driven Architecture (EDA) y Event Sourcing**: [Definición técnica precisa]

Propiedades fundamentales

1. **EDA**: Descripción detallada
2. **Event Sourcing**: Descripción detallada

3. **Eventos como fuente de verdad**: Descripción detallada

4. **Patrón Outbox**: Descripción detallada

Implementación práctica

Conceptos clave

EDA

[Explicación detallada del subtema]

Event Sourcing

[Explicación detallada del subtema]

Eventos como fuente de verdad

[Explicación detallada del subtema]

Patrón Outbox

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Event-Driven Architecture (EDA) y Event Sourcing es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ EDA
2. ✅ Event Sourcing
3. ✅ Eventos como fuente de verdad
4. ✅ Patrón Outbox

EJERCICIOS: TEMA 5.2 - Event-Driven Architecture (EDA) y Event Sourcing

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Event-Driven Architecture (EDA) y Event Sourcing y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Event-Driven Architecture (EDA) y Event Sourcing
2. Analiza las decisiones tomadas

3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Event-Driven Architecture (EDA) y Event Sourcing en un proyecto real

Requisitos:

1. Implementar EDA
2. Implementar Event Sourcing
3. Implementar Eventos como fuente de verdad
4. Implementar Patrón Outbox

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Event-Driven Architecture (EDA) y Event Sourcing para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 5.2 - Event-Driven Architecture (EDA) y Event Sourcing

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Event-Driven Architecture (EDA) y Event Sourcing es correcta?

- [ ] **a)** EDA
- [ ] **b)** Event Sourcing
- [ ] **c)** Eventos como fuente de verdad
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Event-Driven Architecture (EDA) y Event Sourcing siempre mejora el rendimiento del sistema.

- [ ] Verdadero

- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Event-Driven Architecture (EDA) y Event Sourcing en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Event-Driven Architecture (EDA) y Event Sourcing para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. EDA
2. Event Sourcing
3. Eventos como fuente de verdad
4. Patrón Outbox

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Event-Driven Architecture (EDA) y Event Sourcing abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Event-Driven Architecture (EDA) y Event Sourcing no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. EDA
2. Event Sourcing
3. Eventos como fuente de verdad

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Event-Driven Architecture (EDA) y Event Sourcing (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

TEMA 5.3: BROKERS DE MENSAJERÍA

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender brokers de mensajería. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Brokers de Mensajería es un concepto clave en arquitectura de software que...

Definición formal

Brokers de Mensajería: [Definición técnica precisa]

Propiedades fundamentales

1. **Kafka**: Descripción detallada
2. **RabbitMQ**: Descripción detallada
3. **Azure Service Bus**: Descripción detallada
4. **Redis Streams**: Descripción detallada

Implementación práctica

Conceptos clave

Kafka

[Explicación detallada del subtema]

RabbitMQ

[Explicación detallada del subtema]

Azure Service Bus

[Explicación detallada del subtema]

Redis Streams

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Brokers de Mensajería es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ Kafka
2. ✅ RabbitMQ
3. ✅ Azure Service Bus
4. ✅ Redis Streams

EJERCICIOS: TEMA 5.3 - Brokers de Mensajería

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Brokers de Mensajería y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Brokers de Mensajería
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Brokers de Mensajería en un proyecto real

Requisitos:

1. Implementar Kafka
2. Implementar RabbitMQ
3. Implementar Azure Service Bus
4. Implementar Redis Streams

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Brokers de Mensajería para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)

- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 5.3 - Brokers de Mensajería

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Brokers de Mensajería es correcta?

- [ ] **a)** Kafka
- [ ] **b)** RabbitMQ

- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Brokers de Mensajería siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Brokers de Mensajería en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Brokers de Mensajería para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. Kafka
2. RabbitMQ
3. Azure Service Bus
4. Redis Streams

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Brokers de Mensajería abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Brokers de Mensajería no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. Kafka
2. RabbitMQ
3. Azure Service Bus

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Brokers de Mensajería (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

TEMA 5.4: DISEÑO DE CONTRATOS

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender diseño de contratos. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Diseño de Contratos es un concepto clave en arquitectura de software que...

Definición formal

Diseño de Contratos: [Definición técnica precisa]

Propiedades fundamentales

1. **Versionamiento**: Descripción detallada
2. **Backward compatibility**: Descripción detallada
3. **Esquemas (JSON Schema, Protobuf)**: Descripción detallada

Implementación práctica

Conceptos clave

Versionamiento

[Explicación detallada del subtema]

Backward compatibility

[Explicación detallada del subtema]

Esquemas (JSON Schema, Protobuf)

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Diseño de Contratos es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ Versionamiento
2. ✅ Backward compatibility
3. ✅ Esquemas (JSON Schema, Protobuf)

EJERCICIOS: TEMA 5.4 - Diseño de Contratos

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Diseño de Contratos y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Diseño de Contratos
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Diseño de Contratos en un proyecto real

Requisitos:

1. Implementar Versionamiento
2. Implementar Backward compatibility
3. Implementar Esquemas (JSON Schema, Protobuf)

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Diseño de Contratos para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 5.4 - Diseño de Contratos

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Diseño de Contratos es correcta?

- [ ] **a)** Versionamiento
- [ ] **b)** Backward compatibility

- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Diseño de Contratos siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Diseño de Contratos en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Diseño de Contratos para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. Versionamiento
2. Backward compatibility
3. Esquemas (JSON Schema, Protobuf)

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Diseño de Contratos abarca todos estos aspectos porque...

Correcta: Falso

Justificación: Diseño de Contratos no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. Versionamiento
2. Backward compatibility
3. Esquemas (JSON Schema, Protobuf)

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Diseño de Contratos (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

MODULO 6

TEMA 6.1: TEOREMA CAP Y PACELC

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender teorema cap y pacelc. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Teorema CAP y PACELC es un concepto clave en arquitectura de software que...

Definición formal

> **Teorema CAP y PACELC**: [Definición técnica precisa]

Propiedades fundamentales

1. **CAP**: Descripción detallada
2. **PACELC**: Descripción detallada
3. **Implicaciones en bases distribuidas**: Descripción detallada

Implementación práctica

Conceptos clave

CAP

[Explicación detallada del subtema]

PACELC

[Explicación detallada del subtema]

Implicaciones en bases distribuidas

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Teorema CAP y PACELC es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ CAP
2. ✅ PACELC
3. ✅ Implicaciones en bases distribuidas

EJERCICIOS: TEMA 6.1 - Teorema CAP y PACELC

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Teorema CAP y PACELC y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Teorema CAP y PACELC
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Teorema CAP y PACELC en un proyecto real

Requisitos:

1. Implementar CAP
2. Implementar PACELC
3. Implementar Implicaciones en bases distribuidas

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Teorema CAP y PACELC para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 6.1 - Teorema CAP y PACELC

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse

- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Teorema CAP y PACELC es correcta?

- [ ] **a)** CAP
- [ ] **b)** PACELC
- [ ] **c)** Implicaciones en bases distribuidas
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Teorema CAP y PACELC siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Teorema CAP y PACELC en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Teorema CAP y PACELC para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. CAP
2. PACELC
3. Implicaciones en bases distribuidas

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Teorema CAP y PACELC abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Teorema CAP y PACELC no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. CAP
2. PACELC
3. Implicaciones en bases distribuidas

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Teorema CAP y PACELC (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

TEMA 6.2: CONSISTENCIA EVENTUAL VS FUERTE

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender consistencia eventual vs fuerte. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Consistencia Eventual vs Fuerte es un concepto clave en arquitectura de software que...

Definición formal

> **Consistencia Eventual vs Fuerte**: [Definición técnica precisa]

Propiedades fundamentales

1. **Modelos de consistencia**: Descripción detallada
2. **Staleness**: Descripción detallada
3. **Read your own writes**: Descripción detallada

Conceptos clave

Modelos de consistencia

[Explicación detallada del subtema]

Staleness

[Explicación detallada del subtema]

Read your own writes

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Consistencia Eventual vs Fuerte es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ Modelos de consistencia
2. ✅ Staleness
3. ✅ Read your own writes

EJERCICIOS: TEMA 6.2 - Consistencia Eventual vs Fuerte

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Consistencia Eventual vs Fuerte y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Consistencia Eventual vs Fuerte
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Consistencia Eventual vs Fuerte en un proyecto real

Requisitos:

1. Implementar Modelos de consistencia
2. Implementar Staleness
3. Implementar Read your own writes

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐⭐⭐⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Consistencia Eventual vs Fuerte para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 6.2 - Consistencia Eventual vs Fuerte

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Consistencia Eventual vs Fuerte es correcta?

- [ ] **a)** Modelos de consistencia
- [ ] **b)** Staleness
- [ ] **c)** Read your own writes
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Consistencia Eventual vs Fuerte siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Consistencia Eventual vs Fuerte en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Consistencia Eventual vs Fuerte para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. Modelos de consistencia
2. Staleness
3. Read your own writes

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Consistencia Eventual vs Fuerte abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Consistencia Eventual vs Fuerte no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. Modelos de consistencia
2. Staleness
3. Read your own writes

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Consistencia Eventual vs Fuerte (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

TEMA 6.3: PATRONES DE DATOS DISTRIBUIDOS

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender patrones de datos distribuidos. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Patrones de Datos Distribuidos es un concepto clave en arquitectura de software que...

Definición formal

> **Patrones de Datos Distribuidos**: [Definición técnica precisa]

Propiedades fundamentales

1. **CQRS**: Descripción detallada
2. **Database-per-service**: Descripción detallada
3. **Materialized Views**: Descripción detallada
4. **Replicación**: Descripción detallada

Conceptos clave

CQRS

[Explicación detallada del subtema]

Database-per-service

[Explicación detallada del subtema]

Materialized Views

[Explicación detallada del subtema]

Replicación

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Patrones de Datos Distribuidos es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ CQRS
2. ✅ Database-per-service
3. ✅ Materialized Views
4. ✅ Replicación

EJERCICIOS: TEMA 6.3 - Patrones de Datos Distribuidos

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Patrones de Datos Distribuidos y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Patrones de Datos Distribuidos
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Patrones de Datos Distribuidos en un proyecto real

Requisitos:

1. Implementar CQRS
2. Implementar Database-per-service
3. Implementar Materialized Views
4. Implementar Replicación

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios

- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Patrones de Datos Distribuidos para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 6.3 - Patrones de Datos Distribuidos

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Patrones de Datos Distribuidos es correcta?

- [ ] **a)** CQRS
- [ ] **b)** Database-per-service
- [ ] **c)** Materialized Views
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Patrones de Datos Distribuidos siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Patrones de Datos Distribuidos en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Patrones de Datos Distribuidos para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. CQRS
2. Database-per-service
3. Materialized Views

4. Replicación

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Patrones de Datos Distribuidos abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Patrones de Datos Distribuidos no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. CQRS

2. Database-per-service
3. Materialized Views

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Patrones de Datos Distribuidos (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

TEMA 6.4: TRANSACCIONES DISTRIBUIDAS

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender transacciones distribuidas. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Transacciones Distribuidas es un concepto clave en arquitectura de software que...

Definición formal

Transacciones Distribuidas: [Definición técnica precisa]

Propiedades fundamentales

3. **Two-Phase Commit (por qué evitarlo)**: Descripción detallada

Implementación práctica

Conceptos clave

SAGA Pattern

[Explicación detallada del subtema]

Coreografía vs Orquestación

[Explicación detallada del subtema]

Two-Phase Commit (por qué evitarlo)

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Transacciones Distribuidas es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ SAGA Pattern
2. ✅ Coreografía vs Orquestación
3. ✅ Two-Phase Commit (por qué evitarlo)

EJERCICIOS: TEMA 6.4 - Transacciones Distribuidas

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Transacciones Distribuidas y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Transacciones Distribuidas
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Transacciones Distribuidas en un proyecto real

Requisitos:

1. Implementar SAGA Pattern
2. Implementar Coreografía vs Orquestación
3. Implementar Two-Phase Commit (por qué evitarlo)

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Transacciones Distribuidas para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 6.4 - Transacciones Distribuidas

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Transacciones Distribuidas es correcta?

- [ ] **a)** SAGA Pattern
- [ ] **b)** Coreografía vs Orquestación
- [ ] **c)** Two-Phase Commit (por qué evitarlo)
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Transacciones Distribuidas siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Escenario: Un equipo está implementando Transacciones Distribuidas en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Transacciones Distribuidas para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. SAGA Pattern
2. Coreografía vs Orquestación
3. Two-Phase Commit (por qué evitarlo)

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Transacciones Distribuidas abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Transacciones Distribuidas no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. SAGA Pattern

3. Two-Phase Commit (por qué evitarlo)

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Transacciones Distribuidas (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

MODULO 7

TEMA 7.1: CIRCUIT BREAKER Y RETRY PATTERN

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender circuit breaker y retry pattern. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Circuit Breaker y Retry Pattern es un concepto clave en arquitectura de software que...

Definición formal

> **Circuit Breaker y Retry Pattern**: [Definición técnica precisa]

Propiedades fundamentales

1. **Circuit Breaker**: Descripción detallada
2. **Retry Pattern**: Descripción detallada
3. **Timeouts**: Descripción detallada
4. **Bulkheads**: Descripción detallada

Implementación práctica

Conceptos clave

Circuit Breaker

[Explicación detallada del subtema]

Retry Pattern

[Explicación detallada del subtema]

Timeouts

[Explicación detallada del subtema]

Bulkheads

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Circuit Breaker y Retry Pattern es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ Circuit Breaker
2. ✅ Retry Pattern
3. ✅ Timeouts
4. ✅ Bulkheads

EJERCICIOS: TEMA 7.1 - Circuit Breaker y Retry Pattern

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Circuit Breaker y Retry Pattern y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto

- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Circuit Breaker y Retry Pattern
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Circuit Breaker y Retry Pattern en un proyecto real

Requisitos:

1. Implementar Circuit Breaker
2. Implementar Retry Pattern
3. Implementar Timeouts
4. Implementar Bulkheads

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Circuit Breaker y Retry Pattern para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 7.1 - Circuit Breaker y Retry Pattern

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Circuit Breaker y Retry Pattern es correcta?

- [ ] **a)** Circuit Breaker
- [ ] **b)** Retry Pattern
- [ ] **c)** Timeouts
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Circuit Breaker y Retry Pattern siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Circuit Breaker y Retry Pattern en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Circuit Breaker y Retry Pattern para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. Circuit Breaker
2. Retry Pattern
3. Timeouts
4. Bulkheads

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Circuit Breaker y Retry Pattern abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Circuit Breaker y Retry Pattern no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. Circuit Breaker
2. Retry Pattern

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Circuit Breaker y Retry Pattern (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

TEMA 7.2: OBSERVABILIDAD

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender observabilidad. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Observabilidad es un concepto clave en arquitectura de software que...

Definición formal

> **Observabilidad**: [Definición técnica precisa]

Propiedades fundamentales

1. **Logs**: Descripción detallada
2. **Metrics**: Descripción detallada
3. **Tracing**: Descripción detallada

5. **Dashboards**: Descripción detallada

Implementación práctica

Conceptos clave

Logs

[Explicación detallada del subtema]

Metrics

[Explicación detallada del subtema]

Tracing

[Explicación detallada del subtema]

Correlación

[Explicación detallada del subtema]

Dashboards

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Observabilidad es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ Logs
2. ✅ Metrics
3. ✅ Tracing
4. ✅ Correlación
5. ✅ Dashboards

EJERCICIOS: TEMA 7.2 - Observabilidad

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Pregunta: Explica con tus propias palabras qué es Observabilidad y por qué es importante.

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Observabilidad

2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Observabilidad en un proyecto real

Requisitos:

1. Implementar Logs
2. Implementar Metrics
3. Implementar Tracing
4. Implementar Correlación
5. Implementar Dashboards

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Observabilidad para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 7.2 - Observabilidad

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Observabilidad es correcta?

- [ ] **a)** Logs
- [ ] **b)** Metrics
- [ ] **c)** Tracing
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Observabilidad siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Observabilidad en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Observabilidad para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. Logs
2. Metrics
3. Tracing
4. Correlación
5. Dashboards

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Observabilidad abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Observabilidad no siempre mejora el rendimiento. Existen trade-offs como...

Elementos clave a considerar:

1. Logs
2. Metrics
3. Tracing

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Observabilidad (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

TEMA 7.3: ARQUITECTURA SERVERLESS Y CLOUD NATIVE

Tiempo estimado: 45 minutos
Nivel: Intermedio

Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender arquitectura serverless y cloud native. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

Arquitectura Serverless y Cloud Native es un concepto clave en arquitectura de software que...

Definición formal

> **Arquitectura Serverless y Cloud Native**: [Definición técnica precisa]

Propiedades fundamentales

1. **Serverless**: Descripción detallada
2. **Cloud Native**: Descripción detallada
3. **Contenedores**: Descripción detallada
4. **Kubernetes**: Descripción detallada
5. **Autoscaling**: Descripción detallada

Implementación práctica

Conceptos clave

Serverless

[Explicación detallada del subtema]

Cloud Native

[Explicación detallada del subtema]

Contenedores

[Explicación detallada del subtema]

Kubernetes

[Explicación detallada del subtema]

Autoscaling

[Explicación detallada del subtema]

Casos de uso en producción

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: Arquitectura Serverless y Cloud Native es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ Serverless
2. ✅ Cloud Native
3. ✅ Contenedores
4. ✅ Kubernetes
5. ✅ Autoscaling

EJERCICIOS: TEMA 7.3 - Arquitectura Serverless y Cloud Native

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Objetivo: Verificar comprensión de conceptos básicos

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de Arquitectura Serverless y Cloud Native
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar Arquitectura Serverless y Cloud Native en un proyecto real

Requisitos:

1. Implementar Serverless

2. Implementar Cloud Native
3. Implementar Contenedores
4. Implementar Kubernetes
5. Implementar Autoscaling

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Ejercicio 4: Optimización ⭐ ⭐ ⭐ ⭐

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar Arquitectura Serverless y Cloud Native para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 7.3 - Arquitectura Serverless y Cloud Native

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre Arquitectura Serverless y Cloud Native es correcta?

- [ ] **a)** Serverless
- [ ] **b)** Cloud Native
- [ ] **c)** Contenedores
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

Arquitectura Serverless y Cloud Native siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando Arquitectura Serverless y Cloud Native en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice Arquitectura Serverless y Cloud Native para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. Serverless
2. Cloud Native
3. Contenedores

5. Autoscaling

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: Arquitectura Serverless y Cloud Native abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: Arquitectura Serverless y Cloud Native no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. Serverless
2. Cloud Native
3. Contenedores

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de Arquitectura Serverless y Cloud Native (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

Tiempo estimado: 45 minutos
Nivel: Intermedio
Prerrequisitos: Conceptos del módulo anterior

¿Por qué importa este concepto?

Este tema es fundamental para comprender devops para arquitectos. En sistemas de producción, dominar estos conceptos permite tomar decisiones arquitectónicas informadas.

Comprensión intuitiva

DevOps para Arquitectos es un concepto clave en arquitectura de software que...

Definición formal

> **DevOps para Arquitectos**: [Definición técnica precisa]

Propiedades fundamentales

1. **CI/CD**: Descripción detallada
2. **Infraestructura como Código**: Descripción detallada
3. **GitOps**: Descripción detallada

Implementación práctica

Conceptos clave

CI/CD

[Explicación detallada del subtema]

Infraestructura como Código

[Explicación detallada del subtema]

GitOps

[Explicación detallada del subtema]

Aplicación 1: Caso Real

Contexto: Descripción del escenario **Solución**: Cómo se aplica este concepto **Resultado**: Impacto medible

Errores frecuentes

❌ Error 1: [Descripción]

Por qué falla: Explicación técnica **Solución correcta**: Enfoque correcto

Resumen del concepto

En una frase: DevOps para Arquitectos es...

Cuándo usarlo: Criterios de aplicación

Prerequisito crítico: Conceptos que deben dominarse antes

Siguiente paso: Enlace al próximo tema

Puntos clave

1. ✅ CI/CD
2. ✅ Infraestructura como Código
3. ✅ GitOps

EJERCICIOS: TEMA 7.4 - DevOps para Arquitectos

Ejercicios Conceptuales

Ejercicio 1: Comprensión ⭐

Criterios de evaluación:

- [ ] Define correctamente el concepto
- [ ] Identifica casos de uso
- [ ] Explica beneficios y trade-offs

Ejercicio 2: Análisis ⭐ ⭐

Objetivo: Analizar un escenario real

Escenario: [Descripción de un caso práctico]

Tareas:

1. Identifica los elementos clave de DevOps para Arquitectos
2. Analiza las decisiones tomadas
3. Propón mejoras

Solución modelo: [Incluida al final]

Ejercicios Prácticos

Ejercicio 3: Implementación ⭐ ⭐ ⭐

Objetivo: Implementar DevOps para Arquitectos en un proyecto real

Requisitos:

1. Implementar CI/CD
2. Implementar Infraestructura como Código
3. Implementar GitOps

Criterios de aceptación:

- [ ] Código funcional
- [ ] Tests unitarios
- [ ] Documentación

Ejercicio Desafío

Objetivo: Optimizar una implementación existente

Contexto: Sistema con problemas de [aspecto relevante]

Desafío: Aplicar DevOps para Arquitectos para resolver el problema

Rúbrica:

- Análisis del problema (25%)
- Diseño de solución (25%)
- Implementación (30%)
- Testing y validación (20%)

Soluciones Modelo

Solución Ejercicio 2

[Solución detallada con explicación]

Solución Ejercicio 3

```
// Código de ejemplo
```

Explicación: [Rationale de las decisiones]

EVALUACIÓN: TEMA 7.4 - DevOps para Arquitectos

Instrucciones

- Tiempo estimado: 15 minutos
- Todas las preguntas deben responderse
- Consulta el solucionario al final para verificar tus respuestas

---

CUESTIONARIO

Pregunta 1: Opción Múltiple

¿Cuál de las siguientes afirmaciones sobre DevOps para Arquitectos es correcta?

- [ ] **a)** CI/CD
- [ ] **b)** Infraestructura como Código
- [ ] **c)** GitOps
- [ ] **d)** Todas las anteriores

Pregunta 2: Verdadero/Falso

DevOps para Arquitectos siempre mejora el rendimiento del sistema.

- [ ] Verdadero
- [ ] Falso

Justifica tu respuesta:

Pregunta 3: Análisis de Caso

Escenario: Un equipo está implementando DevOps para Arquitectos en su sistema.

Pregunta: ¿Qué consideraciones arquitectónicas deben tener en cuenta?

Tu respuesta:

Pregunta 4: Aplicación Práctica

Diseña una solución que utilice DevOps para Arquitectos para resolver el siguiente problema:

[Descripción del problema]

Elementos que debe incluir tu diseño:

1. CI/CD
2. Infraestructura como Código
3. GitOps

SOLUCIONARIO

Respuesta 1

Correcta: d) Todas las anteriores

Explicación: DevOps para Arquitectos abarca todos estos aspectos porque...

Respuesta 2

Correcta: Falso

Justificación: DevOps para Arquitectos no siempre mejora el rendimiento. Existen trade-offs como...

Respuesta 3

Elementos clave a considerar:

1. CI/CD
2. Infraestructura como Código
3. GitOps

Explicación detallada: [Rationale completo]

Respuesta 4

Solución modelo:

[Diseño completo con diagramas y explicación]

Criterios de evaluación:

- Correcta aplicación de DevOps para Arquitectos (40%)
- Consideración de trade-offs (30%)
- Claridad de la solución (30%)

Tabla de Contenidos