1ASI0729 - DESARROLLO DE APLICACIONES OPEN SOURCE
EXAMEN FINAL
202510

Sección: 4289

Profesores: Bautista Ubillús, Efraín Ricardo
Castro Veramendi, Rafael Oswaldo
Flores Moroco, Juan Antonio
Mori Paiva, Hugo Allan
Robles Fernández, Iván
Sánchez Seña, Alberto Wilmer
Velásquez Núñez, Ángel Augusto

Duración: 170 minutos

Indicaciones:

1. El examen consta de 1 pregunta, y tendrá **170 minutos** para resolverlas.
2. La pregunta es de tipo Proyecto de Software y la entrega de su respuesta es a través de envío de archivo empaquetado **.zip** (único formato válido) con nombre eb<nrc>u<código-estudiante>.zip (por ejemplo, eb4289u201621873.zip), conteniendo el proyecto de software, en la Actividad para el Examen final.
3. Debe crear su proyecto de solución y evidenciar la autoría.
4. Puede utilizar como referencia los materiales publicados en el aula virtual, los sitios web de documentación de frameworks, lenguaje de programación utilizados, así como ejemplos de clase (solo como referencia, no como fuente de duplicado o copia).

Enunciado:

Caso Innovyze

Innovyze (https://innovyze.com/) es un líder mundial en software para la gestión del ciclo de vida del agua, incluyendo modelado hidráulico, análisis de redes de alcantarillado y agua potable, y planificación de infraestructuras críticas. Sus soluciones permiten a las empresas de servicios públicos y municipios optimizar sus operaciones, mejorar la resiliencia de la infraestructura y responder eficazmente a incidentes.

Innovyze está expandiendo su plataforma "AquaResponse" para incluir un módulo avanzado de gestión de incidentes y planes de evacuación. Este módulo tiene como objetivo centralizar la información de reportes de averías, automatizar evaluaciones de riesgo y, en casos críticos, generar órdenes de evacuación integradas con sistemas de alerta temprana.

Actualmente, el equipo de desarrollo está enfocado en la creación de una API de backend robusta para el módulo "IncidentOps", que se encargará de registrar reportes de averías en la infraestructura hídrica, realizar evaluaciones automáticas basadas en criterios predefinidos, y emitir órdenes de evacuación cuando la gravedad del incidente lo requiera.

Pregunta 1 (20 p.).

Usted se integra al backend software developer team, a cargo de la creación de un **RESTful API** que brinde soporte a las operaciones de Innovyze. El ecosistema de la plataforma AquaResponse de Innovyze requiere que se cuenten con **Endpoints** en el RESTful API, para el manejo de la información de:
RiskParameters, conformada por los atributos id, infrastructureType, failureType, minImpactValue, maxImpactValue, severityThreshold.
FailureReports, conformada por id, assetCode, infrastructureType, failureType, reportedImpactValue, state, reportedAt, resolvedAt.
EvacuationOrders, conformada por id, assetCode, affectedZones, reason, priority, issuedAt, state, validUntil.

La tabla asociada a RiskParameters debe contener la siguiente información:

id	infrastructureType	failureType	minImpactValue	maxImpactValue	severityThreshold
1	WATER_PIPE	burst	0.0	100.0	75.0
2	SEWER_LINE	overflow	0.0	50.0	40.0
3	PUMPING_STATION	mechanical_failure	0.0	10.0	8.0
4	RESERVOIR	structural_crack	0.0	200.0	150.0

El valor de state en FailureReport corresponde con un FailureReportState enumeration con los posibles valores:

Id	Name
0	PENDING
1	EVALUATED
2	RESOLVED
3	CLOSED

El valor de state en EvacuationOrder corresponde con un EvacuationOrderState enumeration con los posibles valores:

Id	Name
0	ISSUED
1	ACTIVE
2	COMPLETED
3	CANCELLED

El valor de priority corresponde con un EvacuationOrderPriority enumeration con los posibles valores:

Id	Name
0	LOW
1	MEDIUM
2	HIGH
3	URGENT

Como reglas de negocio, Innovyze:

- Establece que la información que se desea preservar de los *Risk Parameters*, incluye **id** (Long, Primary Key, Autogenerado), **infrastructureType** (String, obligatorio, no vacío), **failureType** (String, obligatorio, no vacío), **minImpactValue** (Double, obligatorio), **maxImpactValue** (Double, obligatorio), **severityThreshold** (Double, obligatorio).

- Establece que la información que se desea preservar de los *Failure Reports*, incluye **id** (Long, Primary Key, Autogenerado), **assetCode** [1] (identificador del negocio, embedded type obligatorio, no vacío, solo puede contener un valor UUID válido), **infrastructureType** (String, obligatorio, no vacío), **failureType** (String, obligatorio, no vacío), **reportedImpactValue** (Double, obligatorio)**, state** (FailureReportState enumeration, obligatorio, no nulo, por defecto PENDING), **reportedAt** (LocalDateTime, obligatorio, no nulo), **resolvedAt** (LocalDateTime, opcional, puede ser nulo, pero si está presente no debe ser anterior a reportedAt).
- Establece que la información que se desea preservar de *Evacuation Orders*, incluye **id** (Long, Primary Key, autogenerado), **assetCode** (identificador del negocio, embedded type obligatorio, no vacío, solo puede contener un valor UUID válido), **affectedZones** (List<String>, obligatorio, no vacía)**, reason** (String, obligatorio, no vacío), **priority** (EvacuationOrderPriority enumeration, obligatorio, no nulo) **issuedAt** (LocalDateTime, obligatorio, no nulo), **state** (EvacuationOrderState enumeration, obligatorio, no nulo, por defecto ISSUED), **validUntil** (LocalDateTime, obligatorio, no nulo, debe ser posterior a issuedAt).
- Requiere que la información de *Risk Parameters* sea poblada en la base de datos de forma automática asociada al ApplicationReady event.
- Especifica que el atributo **assetCode** debe ser embedded y solo puede contener un valor UUID válido, que debe proporcionarse (no autogenerarse) al momento de registrar un failure report.
- Especifica que **assetCode** se considera un identificador interno del negocio. No se debe registrar en la base de datos dos **failure reports** con el mismo valor de **assetCode** y **failureType** en el mismo día.
- Requiere que el valor de **reportedAt** no sea mayor que la fecha y hora actual.
- Requiere que el valor de **issuedAt** no sea mayor que la fecha y hora actual, y **validUntil** debe ser al menos 24 horas posterior a issuedAt.
- Especifica que al momento de registrar un **failure report**, si el reportedImpactValue está fuera del rango entre minImpactValue y maxImpactValue o excede el severityThreshold definido para el infrastructureType y failureType correspondiente en RiskParameters, el failure report se registra con state FLAGGED, y se debe emitir un evento IncidentEvaluatedEvent. En caso contrario, el state es ACCEPTED.
- El Event Handler para el IncidentEvaluatedEvent debe: actualizar el state del FailureReport a EVALUATED. Si el reportedImpactValue excede el severityThreshold por más del 15%, se debe generar una EvacuationOrder para el assetCode, con priority URGENT, issuedAt igual a la fecha y hora actual, validUntil igual a issuedAt + 72 horas, state ISSUED, y affectedZones calculadas en base a un servicio externo (simulado por una lógica simple, por ejemplo, "Zone A", "Zone B", "Zone C" si el impacto es muy alto, "Zone A" si el impacto es moderado). El reason debe ser una descripción concisa del tipo de fallo y el impacto. Si el reportedImpactValue excede el severityThreshold por menos del 15% pero más del 5%, la priority es HIGH, validUntil es issuedAt + 48 horas. En cualquier otro caso de exceder el umbral, la priority es MEDIUM, validUntil es issuedAt + 24 horas.
- Especifica que tanto *FailureReport* como *EvacuationOrder* pertenecen a diferentes bounded contexts, por lo que se necesita un *Anti-Corruption Layer* (*ACL*) para que el bounded context que lo requiera, acceda a capacidades que exponga el otro bounded context como parte de la implementación de una característica requerida.
- Especifica que tanto *FailureReport* como *EvacuationOrder* son Aggregate Roots en sus respectivos bounded contexts, por lo que requiere contar con atributos de auditoría para registrar **createdAt** (fecha y hora de creación de registro) y **updatedAt** (fecha y hora de última actualización de registro), de uso interno y poblados de forma automática por el sistema al momento de las operaciones de creación o actualización.

Durante la etapa de desarrollo, le asignan trabajar en específico sobre dos Endpoints:

/api/v1/failure-reports
/api/v1/evacuation-orders

[1] **assetCode** contiene un identificador típico del dominio de un activo de infraestructura hídrica (como una tubería, estación de bombeo o reservorio). Para efectos de la evaluación, debe tratarse de un valor UUID válido que identifica de forma única a un activo dentro de la infraestructura operada por Innovyze.

Incluya como parte del desarrollo la implementación de todas las reglas de negocio.

Failure Reports Endpoint (/api/v1/failure-reports)
Debe implementar **solo una** operación en el RESTful API: agregar (POST) un **failure report**. Los valores de **id** no se solicita, son autogenerados al momento de almacenar la información. Los valores de **assetCode** que se proporciona como String deben poder convertirse a valores válidos de tipo UUID versión 4[2]. Al agregar se debe retornar en el response el status 201 (created). Incluya en el response un objeto con el *id* generado para el *failure report id* y sus demás atributos, incluyendo el asset*Code* como String. No incluya en el response los atributos de auditoría.

Evacuation Orders Endpoint (/api/v1/evacuation-orders)
Debe implementar **solo una** operación en el RESTful API: obtener (GET) de las **evacuation orders.** Debe retornar el conjunto de evacuations orders almacenados actualmente. Incluya en el response los atributos de auditoría.

Technical constraints

1. Elabore la solución con Java 24, Spring Boot Framework 3.5 como development framework y Spring Data JPA como ORM.
2. Cree su proyecto de software con el nombre ***eb<NRC>u<código-estudiante>*** (por ejemplo, *eb4289u201621873*).
3. La información debe ser persistente en una base de datos relacional (MySQL), en un esquema **innovyze_aquaservice**.
4. Los packages de su solución deben tener como nombre raíz **com.innovyze.aquaservice.platform.u<código-estudiante>** (por ejemplo, com.*innovyze.aquaservice.platform.u201621873*).
5. Considere que el concepto **FailureReport** pertenece al bounded context **incidents**, el concepto **RiskParameter** al bounded context **risks**, y el concepto **EvacuationOrder** al bounded context **emergency**.
6. Aplique buenas prácticas de Arquitectura de Software, enfoque de **Domain-Driven Design**, separación en bounded contexts, **layered architecture** (domain, application, interfaces, infrastructure), patrones de strategic y tactical Domain-Driven Design, patrón CQRS, patrón Anti-Corruption Layer (ACL), principios y patrones de diseño de software orientado a objetos, convenciones de nomenclatura en **inglés**, así como buenas prácticas de nomenclatura en Java (entre ellas Upper-Camel-Case para Clases, Lower-Camel-Case para atributos y métodos) y buenas prácticas para nomenclatura de objetos de Base de Datos (entre ellas snake case, tablas en plural, sin mnemónicos).
7. Aplique reglas de object-relational mapping, implementando en shared un physical naming strategy que establezca de forma automática las convenciones de snake case para objetos de base de datos, así como plural para nombres de tablas.
8. Considere el bounded context **shared** para elementos base comunes/reutilizables que puedan ser aprovechados o extendidos por elementos en otros bounded contexts. El contenido del bounded context shared debe seguir las especificaciones del bounded context *shared* del proyecto de ejemplo *learning center platform* revisado en clase.
9. Utilice minúsculas para los nombres de URL y términos compuestos separados por guión medio (-) para todos los endpoints.
10. Utilice la biblioteca Lombok para el manejo de métodos constructores y de acceso en las clases de Java.
11. Utilice records en vez de clases para almacenamiento de valores inmutables.
12. Para **failure report** y **evacuation order**, incluya atributos de auditoría createdAt y updatedAt con valores poblados de forma automática por Spring Boot al momento de la creación.
13. Utilice el patrón Assembler para el Object Mapping en la sección *transform* en interfaces layer.
14. Documente su código con **JavaDoc** (ver referencias), colocando en inglés información de propósito para principales objetos de programación, así como propósito, parámetros y valor retornado en clases y métodos relevantes. Incluya como parte de la documentación sus nombres y apellidos como valor para @author.
15. Incluya documentación de Endpoints con **OpenAPI**.
16. Considere la gestión de excepciones en la aplicación.
17. Empaquete su solución como un archivo **.zip**. **(único formato válido)** con el nombre ***eb<NRC>u<código-estudiante>.zip*** (por ejemplo, *eb4289u201621873.zip*).
18. Suba su archivo de solución en la Actividad indicada para el Examen final.

NO forma parte del alcance del proyecto:

1. Soporte de CORS.
2. Security.
3. Testing.

[2] **Nota:** Para obtener valores UUID versión 4 válidos, puede utilizar la herramienta en línea ***Online UUID Generator*** (ver sección de referencias).

Rúbrica de calificación

Criterio de Calificación	Sobresaliente (S)	Esperado (E)	Necesita Mejorar (M)	Insuficiente (I)	Calificación
C01. Building y ejecución	Al abrir el proyecto y ordenar la ejecución, ésta se inicia sin problemas. El API es accesible en la ruta indicada.	La aplicación no llega a iniciar y ejecutarse, sin embargo el proceso de building llega a concluir.	Al cargar el proyecto el proceso de building presenta errores y no llega a concluir.	No elabora solución.	
	2.0 puntos	**1.0 punto**	**0.5 puntos**	**0 puntos**	
C02. First Endpoint	El RESTful API expone el endpoint especificado en el enunciado. Se evidencia la funcionalidad de las operaciones solicitadas, proporcionando en cada caso los valores esperados y respondiendo adecuadamente ante las excepciones. Se evidencia la persistencia de los objetos solicitados, cumpliendo la estructura según enunciado, en una tabla nombrada según especificaciones, en base de datos relacional según enunciado en el esquema indicado. Se incluye documentación del Endpoint con OpenAPI.	El RESTful API expone el endpoint especificado en el enunciado. Se evidencia parcialmente la funcionalidad de las operaciones solicitadas, proporcionando en algunos casos los valores esperados y respondiendo de forma parcialmente adecuada ante las excepciones, o se evidencia parcialmente la persistencia en base de datos relacional, o se evidencia parcialmente la persistencia de los objetos solicitados con estructura según enunciado, en una tabla *nombrada según especificaciones* en base de datos relacional según enunciado en el esquema *indicado*.	La aplicación implementa y expone el endpoint especificado en el enunciado, pero no cumple con la ruta especificada o no se puede registrar elementos.	La aplicación no implementa o expone el endpoint solicitado.	
	4.0 puntos	**3.0 puntos**	**1.5 puntos**	**0 puntos**	
C03. Second Items Endpoint	El RESTful API expone el endpoint especificado en el enunciado. Se evidencia la funcionalidad de las operaciones solicitadas, proporcionando en cada caso los valores esperados y respondiendo adecuadamente ante las excepciones. Se evidencia la persistencia de los objetos solicitados, cumpliendo la estructura según enunciado, en una tabla nombrada según especificaciones, en base de datos relacional según enunciado en el esquema indicado. Se incluye documentación del Endpoint con OpenAPI.	El RESTful API expone el endpoint especificado en el enunciado. Se evidencia parcialmente la funcionalidad de las operaciones solicitadas, proporcionando en algunos casos los valores esperados y respondiendo de forma parcialmente adecuada ante las excepciones, o se evidencia parcialmente la persistencia en base de datos relacional, o se evidencia parcialmente la persistencia de los objetos solicitados con estructura según enunciado, en una tabla *nombrada según especificaciones* en base de datos relacional según enunciado en el esquema *indicado*.	La aplicación implementa y expone el endpoint especificado en el enunciado, pero no cumple con la ruta especificada o no se puede registrar elementos.	La aplicación no implementa o expone el endpoint solicitado.	
	4.0 puntos	**3.0 puntos**	**1.5 puntos**	**0 puntos**	
C04. Business Rules	El desarrollo incluye la implementación de reglas de negocio, cubriendo de forma completa las condiciones y escenarios establecidos, siendo éstas ejecutables, con adecuado manejo de excepciones, implementando éstas en las capas más adecuadas, aplicando convenciones y buenas prácticas.	El desarrollo incluye la implementación de la mayoría de reglas de negocio, cubriendo de forma parcial las condiciones y escenarios establecidos, siendo éstas ejecutables, implementándolas en las capas adecuadas en la mayoría de casos, ó aplicando parcialmente convenciones y buenas prácticas.	El desarrollo incluye la implementación de algunas de las reglas de negocio, incumpliendo la mayoría de condiciones y escenarios establecidos, siendo éstas ejecutables, implementándolas en las capas adecuadas en muy pocos casos, ó con poca evidencia de aplicar convenciones y buenas prácticas.	No implementa reglas de negocio, o no cubre escenarios más allá de operaciones CRUD básicas, o éstas no son ejecutables.	
	4.0 puntos	**3.0 puntos**	**1.5 puntos**	**0 puntos**	
C05. Code Organization	El desarrollador organiza el código y los elementos de backend de la solución, aplicando buenas prácticas de Java, Spring Boot Framework y Domain-Driven Design, agrupando los elementos de la solución según convenciones, manteniendo organización de paquetes y carpetas recomendadas por el fabricante y buenas prácticas de la industria de software.	El desarrollador aplica en la mayoría de casos para el backend convenciones, recomendaciones y buenas prácticas de Java, Spring Boot Framework y Domain-Driven Design.	El desarrollador aplica en algunos casos para el backend convenciones, recomendaciones y buenas prácticas de Java, Spring Boot Framework y Domain-Driven Design.	No se evidencia un criterio de organización para los elementos de la solución.	
	2.0 puntos	**1.0 puntos**	**0.5 puntos**	**0 puntos**	
C06. Code Quality	Utiliza para el backend el lenguaje de programación Java La codificación tiene un estilo claro, indentando los bloques de código según los estándares de programación correspondientes al lenguaje, aplicando una lógica consistente en los métodos, condicionales sin escenarios no contemplados, uso adecuado de reutilización de código para evitar redundancia. Aplica patrones de arquitectura y patrones de diseño. Distribuye el código en los niveles correspondientes, asignando lógica de persistencia, lógica de negocio, lógica de control, lógica de mapping y transferencia a las interfaces y clases que corresponden. Cumple de forma completa con los technical constraints.	Utiliza para el backend el lenguaje de programación Java. La codificación es funcional, aplica en la mayoría de casos los estándares de indentación de bloques de código, ó existen algunas ineficiencias en la codificación: redundancia ó inconsistencias en la lógica de programación. Aplica parcialmente patrones de arquitectura y patrones de diseño, o existe en algunas partes una distribución de la lógica en los niveles incorrectos. Cumple con la mayoría de technical constraints.	Utiliza para el backend el lenguaje de programación Java. La codificación es funcional, pero solo aplica algunos de los estándares de indentación de bloques de código, ó existen muchas ineficiencias en la codificación: redundancia ó inconsistencias en la lógica de programación. Aplica algunos patrones de arquitectura y patrones de diseño, o existe en muchos casos una distribución de la lógica en los niveles incorrectos. Cumple con solo algunos de los technical constraints.	No utiliza el lenguaje de programación Java para el backend, ó la codificación es funcional pero no se evidencia aplicación de estándares ó criterios de eficiencia en la codificación, con ausencia de comentarios, ó no aplica patrones de arquitectura ni patrones de diseño, o la codificación no es funcional.	
	3.0 puntos	**2.0 puntos**	**1.0 punto**	**0 puntos**	
C07. Naming Standards	El desarrollador aplica en todos los nombres de objetos de programación y base de datos como paquetes, componentes, interfaces, clases, objetos, variables, constantes, métodos, tablas, columas la nomenclatura en inglés y la nomenclatura estándar para identificadores de clases, objetos, miembros de programación, tablas, columnas, así como los recursos.	El desarrollador aplica en la mayoría de casos la nomenclatura en inglés y la nomenclatura estándar para identificadores de clases, objetos, miembros de programación, así como los recursos.	El desarrollador aplica en muy pocos casos la nomenclatura en inglés y la nomenclatura estándar para identificadores de clases, objetos, miembros de programación, así como los recursos.	El desarrollador no aplica nomenclatura en inglés para los objetos de programación ó recursos.	
	1.0 puntos	**0.5 puntos**	**0.25 puntos**	**0 puntos**	
Total	**20 puntos**	**13.5 puntos**	**6.5 puntos**	**0 puntos**	

Lima, 16 de Julio del 2025

Anexos

Anexos A. Referencias

Comprimir y descomprimir archivos:
https://support.microsoft.com/es-es/windows/comprimir-y-descomprimir-archivos-8d28fa72-f2f9-712f-67df-f80cf89fd4e5

REST API Tutorial:
https://restfulapi.net/

Project Lombok:
https://projectlombok.org/

springdoc-openapi:
https://springdoc.org/

Spring Data JPA - Reference Documentation:
https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference

Class UUID:
https://docs.oracle.com/en/java/javase/22/docs/api/java.base/java/util/UUID.html

Online UUID Generator:
https://www.uuidgenerator.net/version4

How to Write Doc Comments for the Javadoc Tool:
https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html

Formats for date and dateTime in JSON payloads
https://www.geeksforgeeks.org/formats-for-date-and-datetime-in-json-payloads/

Spring Data JPA Auditing:
https://docs.spring.io/spring-data/jpa/reference/auditing.html#auditing.annotations