INGENIERÍA DE SOFTWARE
DESARROLLO DE APLICACIONES OPEN SOURCE (1ASI0729)
PRÁCTICA CALIFICADA 2
202520

Sección: 7380
Profesores: Mori Paiva, Hugo Allan
Duración: 100 minutos
Indicaciones:

1. La práctica calificada consta de 1 pregunta, y tendrá 100 minutos para resolverla.
2. La pregunta es de tipo Proyecto de Software y la entrega de su respuesta es a través de envío de archivo empaquetado **.zip** con nombre *pc2<NRC>u<código-estudiante>.zip*, conteniendo el proyecto de software, en la Actividad PC2 (Práctica calificada 2).

Enunciado:

Caso Random User

Su aliado, Random User (https://randomuser.me/)desea promover el uso de su comunidad de ingeniería de software, expondiendo Fake Users, un RESTful API para desarrolladores que brinda acceso a información de Usuarios falsos aleatorios para software developers en general.

Para ello, le solicita elaborar una aplicación backend que pueda gestionar la información de los Ciudadanos *(citizen).*

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 **Random User**

El ecosistema de **Random User** requiere que la aplicación *de backend* cuente con **Endpoints** en el RESTful API, para el manejo de la información de **citizen** conformadas por los atributos citizenId (Long), name (String), nickName(String), BirthDate(Date), Country(String), profession(String)

Como reglas de negocio:

- No permite registrar dos **citizen** con el mismo name, nickName, country y profession.
- El tipo Country es un value object que encapsula un valor enumerado (enum) que representa el país de procedencia. Los valores permitidos del enum son los solo los 10 países que pertencen a la CONMEBOL.

- Establece que la información que se desea preservar de los citizen incluye **id** (long, **obligatorio**, **autogenerado**, **llave primaria**), name (**String**, obligatorio), **nickName** (**String**, obligatorio), BirthDate (Date, obligatorio), Country (**String**, obligatorio), profession (**String**, obligatorio).
- Considere que citizen es un aggregate root y por lo tanto es auditable, con el fin de tener un registro de las fechas de creación y última actualización.
- En el caso de responses que retorna el API con información de **citizen**, para cada **citizen** se debe incluir **todos sus campos incluyendo los valores de auditoría**

Durante la etapa de desarrollo, le asignan trabajar en específico sobre un Endpoint:

/api/v1/**citizens**

Citizen Endpoint (/api/v1/citizens)
Debe implementar **dos** operaciones en el RESTful API: agregar un **Citizen** (POST) y consultar CITIZEN por nickName. Para el POST, el valor de **id** no se ingresa al momento de agregar, pues son autogenerados al momento de almacenar la información. Ante una adición satisfactoria, retorne el HTTP Status 201 y en el body del response el resource incluyendo la información del citizen y su id generado. En caso de error incluya el HTTP Status correcto según el error, así como el mensaje de error en el body del response. La respuesta de GET debe retornar el HTTP Status 200 de encontrar información y 404 de no hacerlo.

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

Technical constraints

1. Elabore la solución con Java 25, Spring Boot Framework 3.5.8 como development framework y Spring Data JPA como ORM.
2. Cree su proyecto de software con el nombre Nombre su proyecto como *pc2**nrcu**code* donde ***nrc*** es su número de NRC y ***code*** es su código de estudiante en minúsculas (por ejemplo: *pc27380u**201621873***).
3. La información debe ser persistente en una base de datos relacional (MySQL), en un esquema **citizendb**.
4. Los packages de su solución deben tener como nombre raíz **com.citizen.platform.u<código-estudiante>** (por ejemplo, *com.***citizen***.platform.u201621873*).
5. Considere que el concepto **citizen** pertenece al bounded context **citizen**.
6. Considere que el bounded context **shared**.
7. Para validación de valores de entrada puede usar Java Annotations predefinidas de Jakarta Bean Validation o @Pattern con regex.
8. 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, Resource, Assembler, 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).
9. 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.
10. 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.

11. Documente su código con **JavaDoc**, 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.
12. Incluya documentación de **OpenAPI** con Swagger UI, redactando textos en inglés.
13. El puerto de escucha del API en **localhost** debe ser el puerto **8080**.
14. Utilice minúsculas y plural para los nombres de URL para todos los endpoints.
15. De ser necesario, puede utilizar la biblioteca Lombok para el manejo de métodos de acceso en las clases de Java.
16. Todos los textos de mensajes deben ser en inglés.
17. Considere la gestión de excepciones en la aplicación.
18. Incluya en el archivo README.md, la información en inglés de la aplicación, descripción y su información como author.
19. Empaquete su solución como un archivo **.zip**. **(único formato válido)** con el nombre ***pc2<NRC>u<código-estudiante>.zip*** (por ejemplo, *pc27380u201621873.zip*).
20. Suba su archivo de solución en la Actividad indicada para la Práctica calificada 2.

NO forma parte del alcance del proyecto:

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

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 puntos**	**0.5 puntos**	**0 puntos**	
C02. 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 indicada en el esquema indicado.	El RESTful API expone el endpoints 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 indicada 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.	
	6.0 puntos	**4.0 puntos**	**2.0 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 las reglas de negocio, cubriendo de forma parcial las condiciones y escenarios establecidos, siendo éstas ejecutables, implementándolas en las capas adecuadas en la mayoría de los casos, o aplicando parcialmente convenciones y buenas prácticas.	El desarrollo incluye la implementación de algunas de las reglas de negocio, incumpliendo la mayoría de las 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 los 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.	
	3.0 puntos	**2.0 puntos**	**1.0 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 los 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 puntos**	**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 los 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.	
	2.0 puntos	**1.0 puntos**	**0.5 puntos**	**0 puntos**	
Total	**20 puntos**	**13.0 puntos**	**6.5 puntos**	**0 puntos**	

Lima, 22 de noviembre 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/

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

Spring Boot OpenAPI documentation: https://springdoc.org/

Spring Boot – Validation using Hibernate Validator
https://www.geeksforgeeks.org/springboot/spring-boot-validation-using-hibernate-validator/

Guide to Field Validation with Jakarta Validation in Spring
https://agussyahrilmubarok.medium.com/guide-to-field-validation-with-jakarta-validation-in-spring-8c9eca68022e