Obligatorio 1 de Diseño de Aplicaciones II

Se desea desarrollar una aplicación que permita administrar las compras de tickets a conciertos en el Antel Arena. Este sistema es utilizado por personas con diferentes perfiles.
A continuación se describen los requerimientos de la solución para los diferentes perfiles que utilizan la aplicación.

- Un **administrador** puede realizar el mantenimiento (alta, baja y modificación) de los siguientes datos:
  - **Género:** Género músical identificado por un nombre.
  - **Banda:** Registran el nombre, año en el cual se formaron, nombre de los integrantes y el género al cual pertenecen.
  - **Artista Solista:** Registran el nombre del artista, año en el cual comenzó su carrera y género al cual pertenece.
  - **(*) Concierto:** Se registra la fecha del mismo, la cantidad de tickets disponibles para la venta, el costo de los mismos, un nombre para el "tour" y el artista o banda el cual es protagonista.
  - **Usuarios:** Gestiona los usuarios del sistema incluyendo el nombre, apellido, email, contraseña y rol (administrador, vendedor, acomodador, espectador).

- Un **vendedor** puede:
  - **Tickets:** sobre estos nos interesa registrar la venta, sabiendo quien lo compró y generando un identificador único con el que el usuario puede verificar la compra del mismo. El ticket posee un estado, el cual puede ser "comprado" o "utilizado".

- Un **acomodador** puede:
  - **(*) Verificar ticket:** dado un identificador, puede escanear dicho ticket y pasar su estado a "utilizado", evitando que varias personas pasen con el mismo ticket.

- Un **espectador** puede:
  - Sin la necesidad de ingresar al sistema puede ver:
    - Próximos conciertos ordenados cronológicamente.
    - Conciertos entres dos fechas.
    - Detalle de un concierto específico.
    - **(*)** Filtrar bandas y artistas por nombre pudiendo ver el detalle completo incluyendo el listado de sus conciertos.
  - Luego de ingresar al sistema puede:
    - **(*)** Realizar la compra de un ticket.
    - Ver los tickets que compró y para cada uno ver los datos incluyendo su estado.
    - Modificar los datos de su cuenta (nombre, apellido y contraseña)

Requerimientos para grupos de TRES estudiantes

- **(*)** Una vez finalizado un concierto; cada espectador que compró y asistió al mismo (ticket utilizado) puede evaluarlo (**Review**) en cualquier momento.

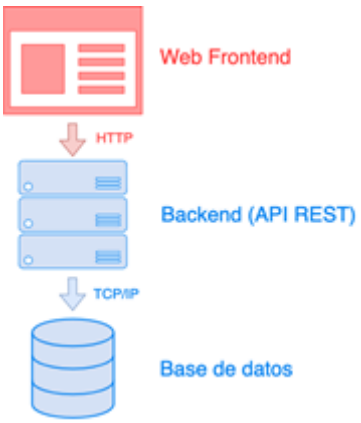
- **(*)** Al loguearse un espectador, el mismo debe ser notificado si tiene conciertos sin evaluar.

- **(*)** El sistema debe contar con un módulo dedicado a la gerencia de la empresa (nuevo rol **gerente**) que permite realizar consultas estadísticas como:

- Los N conciertos más vendidos.
- Análisis de las ventas por género musical entre dos fechas (por defecto el mes corriente). Por ejemplo:

Pop	34%	$ 3,808,000
Rock	29%	$ 3,248,000
Hip Hop	17%	$ 1,904,000
Jazz	11%	$ 1,232,000
Ópera	9%	$ 1,008,000
	---	---
Total	100%	$ 11,200,000

Para resolver el problema se definió la siguiente **arquitectura de alto nivel:**



Artefacto	Descripción
Base de datos	Base de datos relacional (SQL Server) en donde se almacenan los datos de la aplicación.
Api REST + Backend	Toda interacción con el repositorio de datos se realiza mediante un API REST, la que ofrece operaciones para resolver todo lo necesario y el back end donde se implementa la lógica de negocio.
Web Frontend	Aplicación que permite a los usuarios registrarse e interactuar con la aplicación.

Alcance de la primera entrega

Considerando la arquitectura mencionada en la sección anterior, para esta primera **entrega se debe implementar una API REST** que ofrezca todas las operaciones que el alumno considere necesarias para soportar la aplicación descrita en este documento.

Dicho de otra manera, se debe implementar **todo el backend** de la aplicación (lógica de negocio), pero **no el frontend** de la misma (interfaz de usuario). Se espera que, mediante la descripción de la aplicación, y complementando con consultas a aulas cuando considere necesario, el alumno sea capaz de identificar las operaciones que el API debe ofrecer.

Dado que la aplicación no tendrá interfaz de usuario, la forma de interactuar con la misma será mediante un cliente HTTP. Se espera que se utilice **Postman** (https://www.getpostman.com/) como cliente, ya que se debe entregar una exportación de una colección de Postman con los end-points definidos para poder probar. Debe tener todas las llamadas a la API preparadas y utilizar la parametrización de Postman en cuanto a la URL a utilizar, token en el header, etc. para no tener que escribir todas las llamadas nuevamente en la demostración.
En caso de no entregar dicha colección, los docentes pueden optar por la no corrección de la funcionalidad. Ver la rúbrica de evaluación en las secciones más abajo.

Implementación

Se debe entregar una implementación del API REST necesaria para soportar la aplicación que se describe. La entrega debe contener una solución de Visual Studio que agrupe todos los proyectos implementados.
La solución debe incluir el código de las pruebas automáticas. Se requiere escribir los casos de prueba automatizados con MSTests, documentando y justificando las pruebas realizadas.

Se espera que la aplicación se entregue con una base de datos con datos de prueba, de manera de poder comenzar las pruebas sin tener que definir una cantidad de datos iniciales. Dichos datos de prueba deben estar adecuadamente especificados en la documentación entregada.

Tecnologías y herramientas de desarrollo

- Microsoft VSCode/ Visual Studio
- Microsoft SQL Server Express 2017
- Postman
- NET Core SDK (Se recomienda 3.1 o 5.0) / ASP.NET Core (Se recomienda 3.1 o 5.0)(C#)
- Entity Framework Core (Se recomienda 3.1 o 5.0)
- Astah o cualquier otra herramienta UML 2

NOTA: La totalidad y detalle de los requisitos serán relevados a partir de consultas en el foro correspondiente en aulas. Para evitar complejidades innecesarias se realizaron simplificaciones al dominio del problema real.

Independencia de librerías

Se debe diseñar la solución que al modificar el código fuente minimice el impacto del cambio en los componentes físicos de la solución. Debe documentar explícitamente como su solución cumple con este punto. Cada paquete lógico debe ser implementado en un *assembly* independiente, documentando cuáles de los elementos internos al paquete son públicos y cuáles privados, o sea cuáles son las interfaces de cada *assembly*.

Persistencia de los datos

La empresa requiere que todos los datos del sistema sean persistidos en una base de datos. De esta manera, la siguiente vez que se ejecute la aplicación se comenzará con dichos datos cargados con el último estado guardado antes de cerrar la aplicación.

El diseño debe contemplar el modelado de una solución de persistencia adecuada para el problema utilizando Entity Framework (*Code First*).

Se espera que como parte de la entrega se incluya dos respaldos de la base de datos: uno vacío y otro con datos de prueba. Se debe entregar el archivo *.bak* y también el script *.sql* para ambas bases de datos.

Mantenibilidad

La propia empresa eventualmente hará cambios sobre el sistema, por lo que se requiere un alto grado de mantenibilidad, flexibilidad, calidad, claridad del código y documentación adecuada.

Por lo que el desarrollo de todo el obligatorio debe cumplir:

- Estar en un repositorio **Git**.
- Haber sido escrito utilizando **TDD** (desarrollo guiado por pruebas) lo que involucra otras dos prácticas: escribir las pruebas primero (Test First Development) y Refactoring. De esta forma se utilizan las pruebas unitarias para dirigir el diseño.
  Es necesario utilizar **TDD únicamente** para el *back end* y la *API REST*, no para el desarrollo del *front end*.
  Se debe utilizar un framework de Mocking (como Moq, https://www.nuget.org/packages/moq/) para poder realizar pruebas unitarias sobre la lógica de negocio. En caso de necesitar hacer un test double del acceso a datos, podrán hacerlo utilizando el mismo framework de Mocking anterior o de lo contrario con el paquete EF Core InMemory (https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.InMemory).
- Cumplir los lineamientos de **Clean Code** (capítulos 1 al 10, y el 12), utilizando las técnicas y metodologías ágiles presentadas para crear código limpio.

Control de versiones

La gestión del código del obligatorio debe realizarse utilizando UN ÚNICO repositorio Git de **Github**, apoyándose en el flujo de trabajo recomendado por **GitFlow** (https://nvie.com/posts/a-successful-git-branching-model/).

Dicho repositorio debe **pertenecer a la organización de GitHub "ORT-DA2"** (https://github.com/ORT-DA2), en la cual deben estar todos los miembros del equipo.

Al realizar la entrega se debe realizar un *release* en el repositorio y la rama *master* no debe ser afectada luego del hito que corresponde a la entrega del obligatorio.

Evaluación (20 puntos)

Las condiciones de entrega serán evaluadas como si se le estuviese entregando a un cliente real: prolijidad, claridad, profesionalismo, orden, etc.

La entrega debe ser la documentación en formato PDF (incluyendo modelado UML) la cual es subida a gestion.ort.edu.uy (antes de las 21 horas del día de la entrega). En el repositorio Git se debe incluir:

- Una carpeta con el **código** fuente de la aplicación, incluyendo todo lo necesario que permita compilar y ejecutar la aplicación.

- Una carpeta "**Aplicación**" con la aplicación compilada en *release* y todo lo necesario para poder realizar la instalación de la misma (*deploy*).
- Una carpeta "**Documentación**" en la que se incluya la documentación solicitada (incluyendo modelado UML, tener especial cuidado que los diagramas queden legibles en el documento).
- Una carpeta "**Base de datos**" con los archivos *.bak* y *.sql.* Entregar una base de datos vacía y otra con datos de prueba.

La documentación a entregar debe dividirse en **4 documentos**:

- Descripción del diseño (PDF que no debe superar las 20 páginas).
- Evidencia del diseño y especificación de la API.
- Evidencia de la aplicación de TDD y Clean Code (PDF que no debe superar las 15 páginas).
- Evidencia de la ejecución de las pruebas de la API con Postman (PDF que no debe superar las 15 páginas).

Todos los documentos deben cumplir con los siguientes elementos del Documento 302 de la facultad (http://www.ort.edu.uy/fi/pdf/documento302facultaddeingenieria.pdf):

- Capítulo 3, secciones 3.1 (sin la leyenda), 3.2, 3.5, 3.7, 3.8 y 3.9.
- Capítulo 4 (salvo 4.1).
- Capítulo 5.
- Se **debe incluir en las portadas el URL al repositorio del equipo**

Evaluación de	

Descripción del diseño.	El objetivo de este documento es demostrar que el equipo fue capaz de diseñar y documentar el diseño de la solución. La documentación debe ser pensada para que un tercero (corrector) pueda en base a la misma comprender la estructura y los principales mecanismos que están presentes en el código. O sea, debe servir como guía para entender el código y los aspectos más relevantes del diseño y la implementación. El documento **debe** organizarse siguiendo el modelo 4+1 haciendo énfasis en las vistas de diseño e implementación. Este documento no debe incluir paquetes o componentes de los proyectos de prueba. Elementos a evaluar: ● Descripción general del trabajo (qué hace la solución) y errores conocidos (*bugs* o funcionalidades no implementadas). ● Diagrama general de paquetes (namespaces) mostrando los paquetes organizados por capas (*layers*) y sus dependencias. En caso de que haya paquetes anidados, se debe utilizar el conector de *nesting*, mostrando la jerarquía de dichos paquetes. ● Cada paquete debe tener una breve descripción de responsabilidades y un diagrama de clases. ● Descripción de jerarquías de herencia utilizadas (en caso de que así haya sido) ● Modelo de tablas de la estructura de la base de datos. ● Para aquellas funcionalidades que el equipo entienda como relevantes: o Diagramas de interacción que muestran las clases involucradas en estas funcionalidades y las interacciones para lograr la funcionalidad. o Estas interacciones a mostrar pueden ser del flujo a alto nivel, de un cierto algoritmo específico de su obligatorio, etc ● Justificación del diseño explicando mediante texto y diagramas, haciendo énfasis en: o La utilización de mecanismos de inyección de dependencias, fábricas, patrones y principios de diseño, etc. Discutir brevemente cómo estos mecanismos apoyan la mantenibilidad de la aplicación. o Sus propias decisiones propias de diseño que hacen a su obligatorio. Explicación de estructuras, mecanismos	6

	o algoritmos que tienen impacto en la mantenibilidad o desempeño de la solución. ○ Descripción del mecanismo de acceso a datos utilizado. ○ Descripción del manejo de excepciones. ● Diagrama de implementación (componentes) mostrando las dependencias entre los mismos. Justificar el motivo por el cual se dividió la solución en dichos componentes. Se considera como aceptable si: ● Los diagramas presentan el uso adecuado de la notación UML. ● La estructura de la solución representa la descomposición lógica (módulos) y de proyectos de la aplicación. ● Las vistas de módulos, de componentes, modelo de datos y los comportamientos documentados sirven como guía para la comprensión del código implementado. ● La taxonomía de paquetes y clases en los diagramas respeta las convenciones de nombre de C# utilizada en la implementación. ● Se justifica y explica el diseño en base al uso de principios y patrones de diseño. Los mismos se implementan correctamente a partir de su objetivo y teniendo en cuenta sus ventajas y desventajas para favorecer o inhibir la calidad de la solución. El objetivo es describir el impacto de las decisiones tomadas para mejorar la mantenibilidad del sistema. No debe limitarse una descripción de lo realizado. ● Correcto manejo de las excepciones e implementación de acceso a base de datos. ● Se describen claramente los errores conocidos. ● La documentación se encuentra bien organizada, es fácil de leer y su formato corresponde con los ítems indicados del documento 302. ● La documentación no debe superar las 20 páginas.	

Evidencia del diseño y especificación de la API.	Documento describiendo la API conteniendo: ● Discusión de los criterios seguidos para asegurar que la API cumple con los criterios REST. ● Descripción del mecanismo de autenticación de requests. ● Descripción general de códigos de error (1xx, 2xx, 4xx, 3xx, 5xx). ● Descripción de los *resources* de la API. ○ URL base. ○ Para cada *resource* describir: ▪ Resource. ▪ Description. ▪ Endpoints (Verbo + URI). ▪ Parameters. ▪ Responses (para todos los códigos de estado). ▪ Headers. Se considera como aceptable si: ● El diseño de la API REST se basa en buenas prácticas de la industria. Por ejemplo: ○ https://developer.spotify.com/documentation/web-api/reference/ ○ https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design ○ Web API Design: The Missing Link by apigee). ● Para generar la documentación se puede utilizar herramientas como swaggerHub (https://swagger.io/tools/swaggerhub/) que permiten generar la especificación de la API en formato electrónico. ● La documentación se encuentra bien organizada, es fácil de leer y su formato corresponde con los ítems indicados del documento 302.	4.5

Evidencia de Clean Code y de la aplicación de TDD	El trabajo se debe desarrollar en su totalidad siguiendo las prácticas de Clean Code y el enfoque de desarrollo TDD. Con el fin de demostrar la correcta ejecución de TDD, para las funcionalidades marcadas con (*), es necesario demostrar la correcta aplicación de esta técnica. El código debe ajustarse a las prácticas recomendadas por Clean Code. Estas apuntan a que el código sea legible por un tercero. La mejor evidencia de la aplicación de Clean Code es que un tercero (los docentes) puedan leer el código con el menor esfuerzo posible. Resultado de la ejecución de las pruebas. Evidencia del código de pruebas automáticas (unitarias y de integración), reporte de la herramienta de cobertura y análisis del resultado. Como parte de la evaluación se va a revisar el nivel de cobertura de los tests sobre el código entregado, por lo que se debe entregar un reporte y un análisis de la cobertura de las pruebas. Ítems para evaluar para cada una de las funcionalidades indicadas: ● Descripción de la estrategia de TDD seguida (inside – out o outside - in). ● Informe de cobertura para todas las pruebas desarrolladas. ● Es importante mantener una clara separación de los proyectos de prueba de los de la solución. ● **Para las funcionalidades especificadas como prioritarias (*) se debe tener en cuenta:** ○ Dejar evidencia mediante los commits de que se aplicó TDD mostrando la evolución del ciclo de TDD. ○ Análisis de las métricas de cobertura. Se espera que la métrica para las pruebas del código indicado la cobertura se encuentre en un valor entre 90% y 100% de líneas de código. Se debe realizar un análisis de cualquier desvío de estos valores. Se considera como aceptable si: ● El código debe ajustarse a las buenas prácticas de Clean Code. https://www.planetgeek.ch/wp-content/uploads/2013/06/Clean-Code-V2.1.pdf ● Un tercero conocedor de Clean Code pueda leer el código sin dificultad. ● El informe de las pruebas sirve como evidencia de la correcta aplicación de desarrollo guiado por las pruebas (TDD) y técnicas de refactoreo de código. Se justifica claramente el motivo por los cuales se obtuvieron los valores registrados	4.5

	● La documentación se encuentra bien organizada, es fácil de leer y su formato corresponde con los ítems indicados del documento 302. ● La documentación no debe superar las 15 páginas.	
Evidencia de la ejecución de las pruebas de la API con Postman.	Se debe generar colecciones de pruebas en postman **para todas las funcionalidade**s, las cuales el deben incluir en el repositorio. Para las funcionalidades marcadas con (*) **se debe documentar los casos de prueba** elaborados, incluyendo: valores inválidos, valores límites, ingreso de tipos de datos erróneos, datos vacíos, datos nulos, omisión de campos obligatorios, campos redundantes, Body vacío {}, formato de los mensajes inválidos, pruebas de las reglas del negocio como alta de elementos existentes, baja de elementos inexistentes, etc. Se debe entregar un reporte que muestre evidencia del resultado de ejecutar los casos de prueba especificados para la API con Postman para las funcionalidades marcadas con (*). La evidencia se puede registrar realizando un único video publicado en Youtube, en los cual **se pueda apreciar claramente la ejecución de las pruebas**. El link a este video debe incluirse en este documento y se debe verificar que se hayan compartido correctamente y que un tercero pueda verlos. Se considera como aceptable si: ● El documento deja claro que la prueba que se ejecutó y cuál fue el resultado obtenido. ● La documentación se encuentra bien organizada, es fácil de leer y su formato corresponde con los ítems indicados del documento 302. ● La documentación no debe superar las 15 páginas.	5

Información importante

Lectura de obligatorio: 21-03-2022 Plazo máximo de entrega: 09-05-2022

Defensa: A definir por el docente Puntaje mínimo / máximo: 0 / 20 puntos

LA ENTREGA SE REALIZA EN FORMA ONLINE EN ARCHIVO NO MAYOR A 40MB EN FORMATO ZIP, RAR O PDF.

IMPORTANTE:

- Inscribirse.
- Formar grupos de hasta tres personas.
- Subir el trabajo a Gestión antes de la hora indicada, ver hoja al final del documento: “RECORDATORIO”.

RECORDATORIO: IMPORTANTE PARA LA ENTREGA

Obligatorios (Cap.IV.1, Doc. 220)

La entrega de los obligatorios será en formato digital online, a excepción de algunas materias que se entregarán en Bedelía y en ese caso recibirá información específica en el dictado de la misma.

Los principales aspectos a destacar sobre la **entrega online de obligatorios** son:

1. La entrega se realizará desde **gestion.ort.edu.uy**
2. Previo a la conformación de grupos cada estudiante deberá estar inscripto a la evaluación. **Sugerimos realizarlo con anticipación.**
3. **Uno de los integrantes del grupo de obligatorio será el administrador del mismo** y es quien formará el equipo y subirá la entrega
4. Cada equipo debe entregar **un único archivo en formato zip o rar** (los documentos de texto deben ser pdf, y deben ir dentro del zip o rar)
5. El archivo a subir debe tener **un tamaño máximo de 40mb**
6. Les sugerimos **realicen una 'prueba de subida' al menos un día antes**, donde conformarán el'**grupo de obligatorio'.**
7. La **hora tope para subir el archivo será las 21:00** del día fijado para la entrega.
8. La entrega se podrá realizar desde cualquier lugar (ej. hogar del estudiante, laboratorios de la Universidad, etc)
9. Aquellos de ustedes que presenten alguna dificultad con su inscripción o tengan inconvenientes técnicos, por favor pasar por la oficina del Coordinador o por Coordinación adjunta **antes de las 20:00hs.** del día de la entrega.

Si tuvieras una situación particular de fuerza mayor, debes dirigirte con suficiente antelación al plazo de entrega, al Coordinador de Cursos o Secretario Docente.