# Obligatorio 2 de Diseño de Aplicaciones II

Dado que la aplicación para administrar las compras de tickets a conciertos en el Antel Arena tuvo un gran éxito local, se tomó la iniciativa de comercializar internacionalmente, pero con algunas modificaciones para brindar una mejor experiencia a las organizaciones que la usen.

A continuación se describen las mejoras necesarias para que esta aplicación siga siendo un éxito.

- El objetivo principal de esta nueva versión es completar la interfaz de usuario (frontend). Para ello se debe construir una aplicación Angular (SPA) que implemente **TODAS las funcionalidades descritas en la letra de la entrega anterior y en esta**.

- Ahora los conciertos deben contar con la siguiente información adicional, la cual debe ser incluida en el listado de conciertos disponibles para el espectador:
  - País en donde se realizará.
  - Nombre del lugar.
  - Dirección.

- **Nuevo rol de Artista**. Este rol hace referencia a un solista o a cada integrante de una banda (baterista, bajista, cantante, coro, guitarrista, etc).
  El artista debe autenticarse en la aplicación en donde puede ver todos los conciertos que realizó y ver si ese concierto estuvo SOLD OUT (vendió todas las entradas que tenía previsto) o no.

- Ahora las bandas y solistas pueden trabajar en conjunto, esto quiere decir, que un concierto puede ser realizado por una o más bandas o solistas.

- **Importación y exportación de conciertos:** en este proceso de internacionalización se ha identificado la necesidad de importar y exportar los conciertos de sistemas externos de agencias locales de cada país, para lo cual se necesitará nuevos importadores y exportadores pero no se quiere desarrollarlos internamente. En consecuencia, se definió que el sistema pueda tener la opción de agregar nuevos importadores y exportadores desarrollados por terceros de forma dinámica y desde cualquier tipo de fuente pudiendo ser tan diversa como se requiera y formatos. Por ejemplo: desde o hacia un .xml, desde un .csv, desde un .json, leyéndolo/escribiendo de una base de datos, consumiendo un servicio HTTP, etc.

  Como no se conocen todos los formatos posibles y se quiere permitir extensibilidad a futuro, es que se desea que un tercero pueda extender nuestro sistema agregando nuevas formas de importación y/o exportación propias. El equipo debe entonces ofrecer una interfaz mediante la cual un desarrollador pueda poner opciones dentro de un menú de importación, las cuales ejecutarán código desarrollado por terceros. **Estas opciones deben poder ser agregadas al sistema en tiempo de ejecución sin necesidad de recompilar la aplicación.**

  Se debe definir y documentar la/las interfaces, asumiendo que otro desarrollador la va a utilizar

para desarrollar una nueva forma de importar/exportar, explicando claramente cómo utilizarla para dejar importaciones disponibles dentro de la aplicación. Se deben cumplir además con lo siguiente:

- La entrega debe incluir documentación detallada de los mecanismos de extensibilidad incorporados.
- La interfaz gráfica del sistema debe estar diseñada y preparada para trabajar en base a estos mecanismos, de forma de no sufrir cambios para poder agregar nueva funcionalidad.
- Se debe entregar todos los elementos compilados para poder importar y exportar en uno de los siguientes dos formatos (como si fueran hechas por dos desarrolladores externo distintos):
  - Desde un archivo .json.
  - Desde un archivo .xml.
- Estos archivos (json o xml) a importar y exportar se pueden asumir que ya se encuentran en el servidor en alguna ruta específica o se pueden enviar desde el frontend a través del servicio correspondiente. No es necesario enviar el archivo (en binario) a través de una llamada HTTP cliente-servidor.

**Requerimientos para grupos de TRES estudiantes**

- El espectador puede marcar si quiere realizar la compra de las entradas de forma automática (reserva hasta que el administrador confirme la compra) para aquellos conciertos de sus bandas o solistas favoritos.

- Un administrador debe contar con un panel donde se muestran las compras de tickets automáticas y aprobar dichas compras de forma independiente dando así por concretada la compra.

- Las compras o ventas de entradas no estarán disponibles en caso de que hayan entradas automáticas por verificar y se supere el cupo total de entradas disponibles. Por ejemplo:
  - Si hay 10 entradas compradas automáticamente para el concierto X eso quiere decir que un administrador tiene que aprobar dichas compras.
  - Si un espectador o un vendedor intenta de realizar una compra o venta de entradas para el concierto X, esta compra o venta no podrá ser realizada hasta que el administrador haya aprobado las 10 entradas compradas automáticamente.

## Alcance

Se deben realizar todos los cambios solicitados, así como dejar bien documentado el impacto de cada uno de ellos en la implementación (qué paquetes/clases se impactó, qué se agregó o eliminó, qué modificaciones se realizaron, etc.).

**Implementación**

Se mantienen los siguientes requisitos no funcionales de la primera versión:

- La entrega debe contener una solución 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 el mismo framework de pruebas unitarias del primer obligatorio, 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
- NET Core SDK 3.1 o 5 / ASP.NET Core 3 o 5 (utilizando C# como lenguaje)
- Entity Framework Core 3.1.3 o 5
- Astah o cualquier otra herramienta UML 2
- Angular

**Instalación**

El costo de instalación de la aplicación debe de ser mínimo y documentado adecuadamente.

**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*.

**Usabilidad**

La aplicación debe ser fácil de utilizar, intuitiva y atractiva. Se deben tener las consideraciones básicas de usabilidad brindadas en otros cursos (ej: Heurísticas de Nielsen). Se aconseja y fomenta el uso de librerías de terceros como: Bootstrap, Angular Material, Nebular, etc.

**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 sesión o aplicación.

## Diseño: Informe de métricas

Para esta segunda entrega se debe tomar un enfoque de fábrica de software con respecto al diseño utilizado. Esto quiere decir que se **debe promover la mantenibilidad del sistema y analizar al máximo que componentes de la aplicación podrían ser reutilizados** en otras aplicaciones a construir a futuro, y diseñar

e implementar en función de ello, justificando las decisiones tomadas.

**Se pide:** realizar un informe sobre el diseño presentado, **basado en métricas**, que explique los puntos fuertes del diseño y que aspectos podrían ser mejorados. Este informe debe analizar los valores obtenidos de las métricas en función de los principios fundamentales de diseño, analizando cuales se cumplen, cuáles no, que ventajas y desventajas presenta y qué se podría cambiar para mejorar.

Se recomienda utilizar alguna herramienta para calcular las métricas en función del código (por ejemplo, NDepend). Particularmente se espera que se usen las métricas vistas en el curso de: *H, A, I y D (o D')*

## Persistencia en base de datos

Toda la información contenida en el sistema debe ser persistida en una base de datos. El diseño debe contemplar el modelado de una solución de persistencia adecuada para el problema utilizando Entity Framework Core (*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.

Es condición necesaria para obtener el puntaje mínimo del obligatorio, que al menos una entidad del sistema pueda ser persistida.

## 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 *backend* y la *API REST*, no para el desarrollo del *frontend*. Se debe utilizar un framework de Mocking (como 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.
- Por último, 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**. Dicho repositorio debe pertenecer a la organización de GitHub "ORT-DA2" (www.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.**

## Entrega y documentación

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 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) y acceso a los docentes al repositorio Git utilizado por el grupo. En el repositorio Git se debe incluir:

1. Una carpeta con el **código fuente** de la aplicación, incluyendo todo lo necesario que permita compilar y ejecutar la aplicación.
2. Una carpeta **"Aplicación"** con la aplicación compilada en *release* y todo lo necesario para poder realizar la instalación de la misma (*deploy*).
3. 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).
4. 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 debe estar contenida **en un solo documento** teniendo en cuenta los ítems que se describen en el apartado de evaluación. los documentos deben cumplir con los siguientes elementos del Documento 302 de la facultad:

- 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.**
- **Se debe actualizar todos los diagramas que corresponda y agregar los que se considere necesario para mostrar los cambios realizados y justificar las decisiones tomadas. Para esta segunda entrega es necesario repetir la documentación de diseño de la primera entrega actualizada.**

## Evaluación (30 puntos)

| Area | **Demo al cliente** |
|---|---|
| Puntaje | 2 |
| Evaluación (criterios) | Cada equipo deberá realizar una demostración al cliente de su trabajo, **en las computadoras de los laboratorios de la Universidad o de la forma que los docentes comuniquen para el día de la demo**. Dentro de los aspectos que un cliente espera se encuentran:<br>● **Ver la demo cuando él esté listo y no tener que esperar a que el equipo se apronte.** |

| | |
|---|---|
| | ● **Los datos y la funcionalidad a mostrar deben estar preparadas.**<br>● Probar la solución y que la misma funcione sin problemas o con problemas mínimos, y de buena calidad de interfaz de usuario.<br>● Conocer las capacidades de cada uno de los integrantes del equipo, pudiendo preguntar a cualquier integrante sobre la solución, su diseño, el código y sobre cómo fue construida, y así apreciar que fue un trabajo en equipo. Todos los integrantes deben conocer toda la solución.<br>● Verificar el aporte individual al trabajo por parte de cada uno de los integrantes del equipo y en función de los resultados, se podrán otorgar distintas notas a los integrantes del grupo. Se espera que cada uno de los integrantes haya participado en la codificación de parte significativa del obligatorio.<br><br>Esta demostración al cliente hará las veces de **defensa del trabajo**.<br><br>NOTA: El incorrecto funcionamiento de la instalación puede significar la no corrección de la funcionalidad. En el caso de defensa en el laboratorio, durante la defensa cada grupo contará con 15 minutos para la instalación de la aplicación. Luego de transcurridos los mismos se restan puntos al trabajo. |

| Area | **Funcionalidad** |
|---|---|
| Puntaje | 7 |
| Evaluación (criterios) | La asignación de los puntos de este ítem se basará en los siguientes criterios de corrección (escala):<br>● Se implementaron sin errores todos los requerimientos funcionales propuestos.<br>● Se implementaron todos los requerimientos funcionales propuestos, pero se detectan errores menores que no afectan el uso normal del sistema.<br>● Se implementaron los principales requerimientos funcionales, aunque no todos, pero se detectan errores menores que no afectan el uso normal del sistema.<br>● Se implementaron los principales requerimientos funcionales, aunque no todos, pero se detectan errores que afectan el uso normal del sistema.<br>● Los requerimientos funcionales implementados son básicos y/o se detectan errores que afectan el uso normal del sistema. |

| Area | **Diseño y Documentación** |
|---|---|
| Puntaje | 15 |
| Evaluación (criterios) | **El objetivo de este documento es demostrar que el equipo fue capaz de mejorar el diseño realizado y mejorar la documentación de la solución**.<br><br>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.<br><br>El documento debe organizarse siguiendo el modelo 4+1 haciendo énfasis en todas las vistas que conoce. Este documento no debe incluir paquetes o componentes de los proyectos de prueba. |

Elementos a evaluar:

- **La documentación no debe superar las 25 páginas (sin incluir el anexo).**
- 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 asociado.
- 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:
    - Diagramas de interacción que muestran las clases involucradas en estas funcionalidades y las interacciones para lograr la funcionalidad.
    - Estas interacciones a mostrar pueden ser del flujo a alto nivel, de un cierto algoritmo específico de su obligatorio, etc.
- Diagrama de implementación (componentes) mostrando las dependencias entre los mismos. Justificar el motivo por el cual se dividió la solución en dichos componentes.
- Justificación y explicación del diseño en base al uso de principios de diseño, patrones de diseño y métricas.
- Se debe discutir claramente los mecanismos utilizados para permitir la **extensibilidad solicitada en la funcionalidad de este nuevo obligatorio.**
- Se debe explicar claramente el uso de los patrones y principios de diseño utilizados y que valor aportan a la solución **(es importante que la explicación se realice en base a diagramas).**
- **Se debe analizar la calidad del diseño discutiendo la calidad de este en base a las métricas de diseño y contrastándolas respecto a la aplicación de principios.** Es importante discutir en base a las métricas el cumplimento de los principios de: Clausura Común, Reuso común, Abstracciones estables, Dependencias estables y hacer hincapié en las conclusiones que se desprenden de la métrica de distancia, explicando los resultados en base a la abstracción e inestabilidad.
- **Resumen de las mejoras al diseño – se debe describir brevemente las cuatro principales mejoras o cambios que realizó sobre el diseño de la primera entrega** de forma de demostrar cómo la aplicación de principios de diseño y el uso de patrones aportaron para obtener un diseño más mantenible.

**Para el anexo:**

- **Se debe incluir el documento de especificación de la API con sus cambios. Solo con incluir aquellos cambios de esta nueva versión está correcto.**
- **Informe de cobertura analizando la efectividad de las pruebas en base a los valores de cobertura obtenidos.**

La documentación en general 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 componentes<br>● Las vistas de módulos, de componentes, modelo de datos y los comportamientos documentados sirven como guía para la comprensión del código implementado.<br>● La taxonomía de paquetes y clases en los diagramas respeta las convenciones de nombre de C# utilizada en la implementación.<br>● **Se justifica y explica el diseño en base al uso de principios a nivel de clase y paquetes/componentes y patrones de diseño**.<br>● **Se explica correctamente los patrones de diseño utilizados en la solución en base a diagramas de estructura y comportamiento.**<br>● **Se analiza el diseño en base a principios y métricas y se justifican los desvíos obtenidos respecto a los valores de referencia.**<br>● **Se describen claramente las principales mejoras realizadas sobre el diseño de la primera entrega.**<br>● **Se analiza la cobertura lograda por las pruebas discutiendo en forma crítica la efectividad de las pruebas automatizadas.**<br>● Correcto manejo de las excepciones e implementación de acceso a base de datos.<br>● Se describen claramente los errores conocidos.<br>● **Se documentan las mejoras a la API en caso de haber realizado cambios.**<br>● La documentación se encuentra bien organizada, es fácil de leer y su formato corresponde con los ítems indicados del documento 302. |

| Area | **Implementación** |
|---|---|
| Puntaje | 6 |
| Evaluación (criterios) | Se espera ver:<br>● Correcta aplicación de desarrollo guiado por las pruebas (TDD) y técnicas de refactoring de código.<br>● Utilización de buenas prácticas de estilo y codificación y su impacto en la mantenibilidad (Clean Code).<br>● Correcto uso de las tecnologías.<br>● Claridad del código respetando las guías de estilo de C#.<br>● Concordancia con el diseño documentado.<br>● Correcto manejo de excepciones.<br>● Implementación de acceso a base de datos. |

**Información importante**

Lectura de obligatorio: 09-May-2022 Plazo máximo de entrega: 16-Jun-2022
Defensa: A definir por el docente Puntaje mínimo / máximo: 10 / 30 puntos

Los grupos de obligatorio se forman como máximo por 2 estudiantes.
Todas las entregas se realizan mediante las reglas de la entrega electrónica de obligatorios.

*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 dos 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:

5. La entrega se realizará desde **gestion.ort.edu.uy**
6. Previo a la conformación de grupos cada estudiante deberá estar inscripto a la evaluación. **Sugerimos realizarlo con anticipación.**
7. **Uno de los integrantes del grupo de obligatorio será el administrador del mismo** y es quien formará el equipo y subirá la entrega
8. 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)
9. El archivo a subir debe tener **un tamaño máximo de 40mb**
10. Les sugerimos **realicen una 'prueba de subida' al menos un día antes**, donde conformarán el **'grupo de obligatorio'.**
11. La **hora tope para subir el archivo será las 21:00** del día fijado para la entrega.
12. La entrega se podrá realizar desde cualquier lugar (ej. hogar del estudiante, laboratorios de la Universidad, etc)
13. 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.