Obligatorio 2 de Diseño de Aplicaciones II

Basados en el trabajo realizado para la primera entrega, se debe completar la aplicación teniendo en cuenta los siguientes requerimientos.

Nuevas funcionalidades y modificación de requerimientos

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

- Se quiere expandir el sistema a otros tipos de deportes por lo que un encuentro se puede dar entre únicamente dos equipos (por ejemplo, fútbol, básquetbol, vóleibol) o varios atletas (por ejemplo, atletismo, arquería, golf). Los datos de los equipos y atletas son lo mismo, pero el sistema debe permitir encuentros con dos o más participantes.
  El deporte se configura para que sus encuentros sean todos de dos participantes o de mas de dos, debiendo verificar esto al crear los encuentros.

- Un encuentro debe contar con un resultado. Existen dos alternativas para ingresar el resultado de un encuentro:
  - En caso que el encuentro tenga exactamente dos participantes, entonces el resultado se ingresa de forma de indicar quién ganó o si fue empate.
  - En los casos que el encuentro tenga más de dos participantes, entonces se ingresa la posición de cada uno.

- Un usuario puede seleccionar un deporte y ver la tabla de posiciones. La posición se obtiene de los resultados ingresados para los encuentros, se recorren todos los encuentros para ese deporte sumando los resultados de forma de obtener la posición.
  - En caso que el encuentro tenga exactamente dos participantes, entonces se recorren todos los encuentros para ese deporte sumando los resultados (3 puntos por ganar, 1 por empatar y 0 por perder).
  - En los casos que el encuentro tenga más de dos participantes, se recorren todos los encuentros para ese deporte sumando los resultados (3 puntos por primer puesto, 2 puntos por segundo puesto, y 1 punto por tercer puesto).

- Registro de acciones del sistema.
  Con fines de auditoría, el sistema debe mantener un log de las siguientes acciones realizadas por los usuarios:
  - entradas de los usuarios a la aplicación (*login*).
  - ejecución de algoritmos de armado de *fixture*.

  Para cada acción en el log se requiere guardar:
  - tipo de acción (login | fixture)

  - fecha y hora
  - nombre de usuario

  Se espera que el sistema tenga independencia del sistema de log teniendo en cuenta que en un futuro puede surgir una nueva forma de mantener el log. Se espera extensibilidad en tiempo de compilación no en tiempo de ejecución para este requerimiento.

- El administrador cuenta con un nuevo reporte:
  - Listar el log de acciones en el sistema de los usuarios entre dos fechas.

- Se quiere permitir mayor extensibilidad a futuro para la generación de encuentros (*fixture*), se desea que un tercero pueda extender nuestro sistema agregando nuevas formas de generar *fixtures*. La aplicación debe entonces ofrecer una interface mediante la cual un desarrollador pueda poner opciones dentro de un menú de *fixtures*, y las que ejecutarán código desarrollado por terceros. Estas opciones deben poder ser agregadas al sistema sin necesidad de recompilar la aplicación.
  Se debe definir y documentar la interface, asumiendo que otro desarrollador la va a utilizar para desarrollar una nueva forma de generar encuentros, explicando claramente cómo utilizarla para dejar nuevos algoritmos de *fixtures* disponibles dentro de la aplicación.
  - 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 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 Visual Studio Code (lenguaje C#)
- Microsoft SQL Server Express 2014
- Entity Framework Core
- NET Core SDK 2.1.4 / ASP.NET Core 2
- Angular

- Astah o cualquier otra herramienta UML 2

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*.

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.

Usabilidad
La aplicación debe ser fácil de utilizar, intuitiva y atractiva.

Diseño

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).

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 (*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 recomienda entregar el archivo *.bak* y también el script *.sql* para ambas base 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 *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** (nvie.com/posts/a-successful-git-branching-model). 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.**

Documentación

La documentación entregada debe ser en **un solo** documento digital en formato PDF, que contenga la siguiente información ordenada e indexada:

- La documentación no debe superar las 25 páginas pudiéndose complementar mediante la utilización de anexos, por ejemplo, utilizando anexos para el informe de clean code, informe de pruebas y mayor detalle o diagramas para casos puntuales.
- Descripción general del trabajo (1/2 carilla): si alguna funcionalidad no fue implementada o si hay algún

error conocido (*bug*) deben ser descritos aquí.

- Descripción del diseño propuesto, incluyendo los diagramas de paquetes, clases (al menos uno por paquete), interacción para especificar los principales comportamientos del sistema, de componentes y de entrega.
- Justificación de las decisiones de diseño tomadas, explicando los mecanismos generales, aplicación de principios, patrones, mecanismo de persistencia, etc.
- Modelo de tablas de la estructura de la base de datos.
- **Informe basado en métricas sobre el diseño, con fortalezas y oportunidades de mejora.**
- Justificación de Clean Code. Explicación de por qué entiende que su código se puede considerar limpio, y en caso de existir, aclaración de oportunidades de mejora o aspectos en los que cree que no cumple con el estándar.
- 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.
- El documento debe cumplir con los siguientes elementos del Documento 302 de la facultad (http://www.ort.edu.uy/fi/pdf/documento302facultaddeingenieria.pdf)
    a. Capítulo 3, secciones 3.1 (sin la leyenda), 3.2, 3.5, 3.7, 3.8 y 3.9
    b. Capítulos 4 (salvo 4.1) y 5
- **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.**

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

La entrega debe ser la documentación en formato PDF (incluyendo modelado UML) y acceso a los docentes al repositorio Git utilizado por el grupo. En el repositorio Git se debe incluir:

- Una carpeta con la aplicación compilada en *release* para realizar la instalación de la misma.
- Código fuente de la aplicación, incluyendo todo lo necesario que permita compilar y ejecutar la aplicación.
- Una carpeta Documentación en la que se incluya el documento en formato PDF (incluyendo modelado UML, tener especial cuidado que los diagramas queden legibles en el documento).
- Base de datos: entregar una base de datos vacía y otra con datos de prueba.

Evaluación (25 puntos)

	Evaluación de	Puntos

Demo al cliente	Cada equipo deberá realizar una demostración al cliente de su trabajo, **en las computadoras de los laboratorios de ORT**. Dentro de los aspectos que un cliente espera se encuentran: ● **Ver la demo cuando él esté listo y no tener que esperar a que el equipo se apronte.** ● Probar la solución y que la misma funcione sin problemas o con problemas mínimos, y de buena calidad de interfaz de usuario. ● 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. ● 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. Esta demostración al cliente hará las veces de **defensa del trabajo**. 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.	2
Funcionalidad	La asignación de los puntos de este ítem se basará en los siguientes criterios de corrección: • Se implementaron sin errores todos los requerimientos funcionales propuestos. • Se implementaron todos los requerimientos funcionales propuestos, pero se detectan errores menores que no afectan el uso normal del sistema. • Se implementaron los principales requerimientos funcionales, aunque no todos, pero se detectan errores menores que no afectan el uso normal del sistema. • Se implementaron los principales requerimientos funcionales, aunque no todos, pero se detectan errores que afectan el uso normal del sistema. • Los requerimientos funcionales implementados son básicos y/o se detectan errores que afectan el uso normal del sistema.	6
Diseño y documentación	La documentación cumple con lo detallado en la sección documentación. Se considera un diseño y documentación 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 nombres de C# utilizada en la implementación. • Se justifica y explica el diseño en base al uso de principios y patrones de	10

	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 diseño de la API REST se basa en buenas prácticas de la industria (por ejemplo https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design y Web API Design: The Missing Link by apigee). • El informe de Clean Code justifica por qué su código se puede considerar limpio. • El informe de las pruebas sirve como evidencia de la correcta aplicación de las mismas. • Informe de métricas. • 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.	
Implementación	Correcta aplicación de desarrollo guiado por las pruebas (TDD) y técnicas de refactoreo de código. Utilización de buenas prácticas de estilo y codificación y su impacto en la mantenibilidad (Clean Code). Correcto uso de las tecnologías. Claridad del código respetando las guías de estilo de C#. Concordancia con el diseño documentado. Correcto manejo de excepciones. Implementación de acceso a base de datos.	7

Información importante

Lectura de obligatorio: 10-10-2018 Plazo máximo de entrega: 22-11-2018
Defensa: A definir por el docente Puntaje mínimo / máximo: 10 / 25 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 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:

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. S**ugerimos 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.