# Obligatorio 1 de Diseño de Aplicaciones II

El **Ministerio de Turismo** a través de su conocida marca **"Uruguay Natural**" se encuentra trabajando en un plan de remodelación tecnológica de algunas de sus plataformas, con el fin de poder aprovechar la pausa en la actividad turística ocasionada por la pandemia, pudiendo así retomar la misma en un futuro potenciando la llegada de nuevos turistas tanto nacionales como internacionales.

Particularmente, luego de varios meses de trabajo detectaron algunos problemas con su sitio web actual, siendo el principal de ellos que el contenido es estático y demasiado genérico, por lo que es difícil poder atrapar los potenciales turistas sin ofrecerles una propuesta más atractiva y completa. En consecuencia, se tomó la decisión de permitir que los posibles turistas tengan una experiencia end-to-end: desde poder explorar lugares turísticos, hasta evaluar y reservar potenciales paquetes turísticos para cada cliente. Para ello, se decidió crear desde cero el sistema, teniendo foco en los siguientes requerimientos funcionales que son los que mayor valor aportan al negocio:

1) **Búsqueda de puntos turísticos por región y por categoría:**

   Los usuarios (de aquí en más serán denominados también **turistas**), pueden acceder al sitio web y explorar los diferentes lugares a visitar según la región de nuestro país y a través diferentes categorías bien definidas.

   Particularmente, se definen las siguientes **regiones** para representar nuestro país que se saben que no cambiarán:
   - Región metropolitana
   - Región Centro Sur
   - Región Este
   - Región Litoral Norte
   - Región "Corredor Pájaros Pintados"

   Además, dentro de cada región se representan los denominados **puntos turísticos,** pudiendo cada uno pertenecer a diferentes **categorías** (por lo menos una). Estas categorías son:
   - "Ciudades"
   - "Pueblos"
   - "Áreas protegidas"
   - "Playas"
   - Etc.

   *No se espera realizar mantenimiento de regiones y categorías para esta versión de la aplicación, pero si su diseño de entities para ser soportados a nivel de base de datos. Sí se espera poder*

*mantener los puntos turísticos asociados a cada región y categorías lo cual se describe a continuación.*

Por último, cada **punto turístico** tiene un nombre, una descripción de máximo 2000 caracteres y una imagen asociada. Por ejemplo, dado el punto turístico "Punta del Este":

**Nombre**: Punta del Este
**Descripción**: *"Donde el lujo y la naturaleza convergen: Punta del Este es reconocido internacionalmente como uno de los principales balnearios de América y el más exclusivo de la región. Se ubica en el departamento de Maldonado, a sólo una hora y media de la capital del país. Lujosas residencias de veraneo, altos edificios de apartamentos frente al mar, enormes yates en el puerto, hoteles y restaurantes de lujo lo transforman en el balneario de mayor glamour de América".*
**Región:** Región Este
**Categoría**: Ciudades, Playas, Áreas protegidas
**Imagen**: ...

En resumen: una vez los usuarios acceden al sitio, se les lista las cinco regiones disponibles y al elegir una de ellas se les muestran los diferentes puntos turísticos asociados a dicha región, donde se ve el nombre, imagen y categorías de cada punto turístico en un listado y donde el usuario tiene la opción de filtrar por una o varias categorías.

**2) Elegir un punto turístico y realizar una búsqueda de hospedajes**

Los turistas, dado un cierto punto turístico que hayan seleccionado previamente pueden realizar una búsqueda de hospedajes. Para realizar dicha búsqueda se debe ingresar: el punto turístico previamente seleccionado, la fecha de check-in, fecha de check-out y la cantidad de huéspedes. Este último valor debe distinguir según la edad de los huéspedes, siguiendo las siguientes categorías:

- Cantidad de Adultos (13 años o más)
- Cantidad de Niños (de 2 a 12 años)
- Cantidad de Bebés (menos de 2 años)

Una vez ingresada la búsqueda, el usuario obtiene una lista de **hospedajes** que coinciden con la búsqueda previamente ingresada. Cada hospedaje tiene un nombre, cantidad de estrellas (1 a 5), punto turístico al cual pertenece, una dirección, una o varias imágenes, el precio por noche, el precio total para el período seleccionado y una descripción de las características y servicios ofrecidos por el hospedaje. Para calcular el precio total para el período se asume que todos los días

tienen el mismo costo y que todos los niños siempre pagan un 50% y los bebés un 25% del valor del día asociado. Esto aplica a cualquier hospedaje ofrecido por Uruguay Natural.

Por ejemplo, veamos la siguiente búsqueda y sus resultados:

**Punto turístico:** Colonia del Sacramento
**Check-in:** 01/12/2020
**Check-out:** 08/12/2020
**Cantidad de huéspedes:** 4 (dos adultos, un niño y un bebé)

**Resultado: 2 hospedajes**

1. Hospedaje 1:
    - **Nombre**: Radisson Colonia Del Sacramento Hotel
    - **Cantidad estrellas:** 5
    - **Dirección:** Washington Barbot 283, 70000 Colonia del Sacramento, Uruguay
    - **Descripción:** *Este hotel se encuentra a solo 1 calle del centro histórico de Colonia del Sacramento y ofrece vistas panorámicas a la bahía, una magnífica piscina climatizada y bañera de hidromasaje. También hay sauna y bañeras de hidromasaje modernas, WiFi gratuita y aparcamiento. Las habitaciones del Radisson Colonia Del Sacramento son amplias y disponen de TV LCD por cable, minibar y aire acondicionado. Algunas cuentan con balcón y gozan de vistas panorámicas al río de la Plata…*
    - **Imágenes**: …
    - **Precio por noche: $100**
    - **Precio total: $1925**

        *Esto equivale al siguiente cálculo: 2 Adultos (7 * 100 * 2), más 1 niño (7 * 100 * 0.5), más 1 bebé (7 * 100 * 0.25). 7 equivale al número de noches del período.*
2. Hospedaje 2:
    - **Nombre**: Hotel Italiano
    - **Cantidad estrellas:** 3
    - **Dirección:** Intendente Suarez 105 Esq. Manuel Lobo, 70000 Colonia del Sacramento, Uruguay
    - **Descripción:** *El Hotel Italiano ofrece habitaciones con vistas al jardín y WiFi gratuita, además de piscina al aire libre y bañera de hidromasaje, ambas rodeadas por sombrillas de paja y tumbonas. El barrio histórico de Colonia se encuentra a 5 minutos a pie. El hotel Italiano cuenta con gimnasio y piscina cubierta climatizada. También ofrece servicio de masajes. Las habitaciones presentan una decoración sencilla y elegante. Disponen de aire acondicionado.*

- **Imágenes**: …
- **Precio por noche: $40**
- **Precio total: $770**

  *Esto equivale al siguiente cálculo: 2 Adultos (7 * 40 * 2), más 1 niño (7 * 40 * 0.5), más 1 bebé (7 * 40 * 0.25). 7 equivale al número de noches del período.*

Además, es importante filtrar aquellos hospedajes que ya tengan su capacidad completa para el período y cantidad de personas seleccionadas. Para ello, se procederá a describir cómo funciona la capacidad de un hospedaje:

Cada hospedaje tiene una capacidad total (que es un valor interno que es manejado por cada hospedaje y que no es mantenido en este sistema). Una vez la capacidad para un hospedaje es colmada, los administradores son informados y deben marcar a dicho hospedaje como "sin capacidad". Es importante reiterar que en el sistema no se debe manejar nada referente a la capacidad de un hotel o las habitaciones en sí, solo se debe marcar cuando un hotel está sin capacidad forma manual por los administradores.

En caso que un hotel esté sin capacidad, no se pueden realizar más reservas en el hospedaje a través del sitio y el mismo no deberá aparecer en los resultados de búsqueda.

**3) <u>Dado un hospedaje, realizar una reserva:</u>**

Los turistas pueden confirmar una reserva en un hospedaje y los datos de reserva que automáticamente se llenan con los datos de búsqueda (check-in, check-out, cantidad de huéspedes) ingresando su <u>nombre</u>, <u>apellido</u> y el <u>e-mail</u>. Se deja de lado la posible utilización de mecanismos de pago para simplificar esta primera entrega.

Una vez realizada la reserva, estos reciben un <u>código único</u> que identifica la reserva, además de un <u>número telefónico</u> y un <u>texto de información</u> del contacto.

Se pide que los usuarios (**turistas**) puedan entonces:

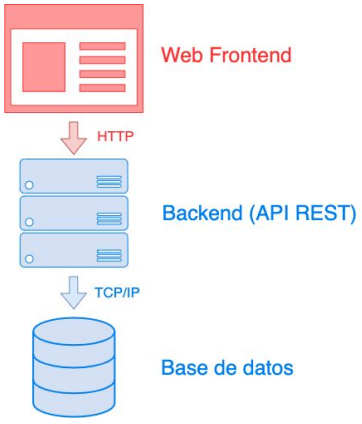
- Buscar **puntos turísticos por región** y filtrar dicha búsqueda **por categoría**.
- **(*)** Buscar **hospedajes** para un cierto **punto turístico** con los parámetros especificados.
- **(*) Realizar una reserva de un hospedaje**.
- **Consultar el estado actual de una reserva** dado su número (el estado actual de una reserva consta de un texto que identifica el estado -su nombre- y una descripción).

- Los estados posibles son [“Creada”, “Pendiente Pago”, “Aceptada”, “Rechazada”, “Expirada”].

A su vez, se cuenta con usuarios **administradores**, que pueden:

- **Iniciar sesión en el sistema** usando su e-mail y contraseña.
- **Dar de alta un nuevo punto turístico,** para una región existente.
- **(*) Dar de alta un nuevo hospedaje** o **borrar uno existente,** para un punto turístico existente.
- **(*) Modificar la capacidad actual de un hospedaje**.
- **(*) Cambiar el estado de una reserva**, indicando una descripción.
- **Un administrador también puede realizar el mantenimiento de los administradores del sistema** (estos tienen nombre, e-mail y contraseña). El e-mail es único y no puede estar repetido.
- Estas operaciones SOLO pueden ser llevadas a cabo por un administrador. Considere inicialmente agregar un usuario super-administrador en base de datos para poder crear futuros administradores.

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 Visual Studio Code (lenguaje C#)
- Microsoft SQL Server Express 2017
- Postman
- NET Core SDK 3.1 / ASP.NET Core 3
- Entity Framework Core 3.1.3
- 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.

## 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 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 *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.**

## 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) y acceso a los docentes al repositorio Git utilizado por el grupo. 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 formato PDF**:

- **Descripción del diseño.**
- **Evidencia del diseño y especificación de la API.**
- **Evidencia de la aplicación de TDD y Clean Code.**
- **Evidencia de la ejecución de las pruebas de la API con Postman.**

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 | | Puntos |
|---|---|---|
| **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.<br><br>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.<br><br>Elementos a evaluar:<br>• Descripción general del trabajo (qué hace la solución) y errores conocidos (*bugs* o funcionalidades no implementadas).<br>• Diagrama de descomposición de los namespaces del proyecto (utilizando el conector nesting) sin dependencias. Solo se debe mostrar la organización y jerarquía de paquetes (namespaces).<br>• Diagrama general de paquetes (namespaces) mostrando los paquetes organizados por capas (*layers*) y sus dependencias.<br>• Para cada paquete breve descripción de responsabilidades y un diagrama de clases.<br>• Descripción de jerarquías de herencia utilizadas.<br>• Modelo de tablas de la estructura de la base de datos.<br>• Para las funcionalidades claves de la solución:<br>o Colaboraciones que muestran las clases involucradas en estas funcionalidades.<br>o Diagramas de secuencia de que muestren las interacciones para lograr la funcionalidad.<br>• Justificación del diseño explicando mediante texto y diagramas, | 6 |

| | | |
|---|---|---|
| | haciendo énfasis en:<br>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.<br>o Explicación de mecanismos o algoritmos que tienen impacto en la mantenibilidad o desempeño de la solución.<br>o Descripción del mecanismo de acceso a datos utilizado.<br>o Descripción del manejo de excepciones.<br>● Diagrama de implementación (componentes) mostrando las dependencias entre los mismos. Justificar el motivo por el cual se dividió la solución en dichos componentes.<br><br>Se considera como aceptable si:<br>● Los diagramas presentan el uso adecuado de la notación UML.<br>● La estructura de la solución representa la descomposición lógica (módulos) y de proyectos de la aplicación.<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 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.<br>● Correcto manejo de las excepciones e implementación de acceso a base de datos.<br>● Se describen claramente los errores conocidos.<br>● La documentación se encuentra bien organizada, es fácil de leer y su formato corresponde con los ítems indicados del documento 302.<br>● La documentación no debe superar las 20 páginas. | |
| **Evidencia del diseño y especificación de la API.** | Documento describiendo la API conteniendo:<br>● Discusión de los criterios seguidos para asegurar que la API cumple con los criterios REST.<br>● Descripción del mecanismo de autenticación de *requests*.<br>● Descripción general de códigos de error (1xx, 2xx, 4xx, 3xx, 5xx).<br>● Descripción de los *resources* de la API.<br>o URL base. | 4.5 |

| | | |
|---|---|---|
| | ○ Para cada *resource* describir:<br>▪ Resource.<br>▪ Description.<br>▪ Endpoints (Verbo + URI).<br>▪ Parameters.<br>▪ Responses (para todos los códigos de estado).<br>▪ Headers.<br><br>Se considera como aceptable si:<br>● 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 y Web API Design: The Missing Link by apigee).<br>● 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.<br>● La documentación se encuentra bien organizada, es fácil de leer y su formato corresponde con los ítems indicados del documento 302. | |
| **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.<br><br>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.<br><br>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. | 4.5 |

| | | |
|---|---|---|
| | Ítems para evaluar para cada una de las funcionalidades indicadas:<br>• Descripción de la estrategia de TDD seguida (inside – out o outside - in).<br>• Informe de cobertura para todas las pruebas desarrolladas.<br>• Es importante mantener una clara separación de los proyectos de prueba de los de la solución.<br>• **Para las funcionalidades especificadas como prioritarias (*) se debe tener en cuenta:**<br>o Dejar evidencia mediante los commits de que se aplicó TDD mostrando la evolución del ciclo de TDD.<br>o 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.<br><br>Se considera como aceptable si:<br>• 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<br>• Un tercero conocedor de Clean Code pueda leer el código sin dificultad.<br>• 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<br>• La documentación se encuentra bien organizada, es fácil de leer y su formato corresponde con los ítems indicados del documento 302.<br>• 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 funcionalidades**, las cuales el deben incluir en el repositorio.<br><br>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. | 5 |

| | | |
|---|---|---|
| | 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 (*).<br><br>La evidencia se puede registrar mediante capturas de pantalla o realizando uno o más videos cortos publicados en Youtube, en los cuales **se pueda apreciar claramente la ejecución de las pruebas**. Los links a estos videos deben incluirse en este documento y se debe verificar que se hayan compartido correctamente y que un tercero pueda verlos.<br><br>Se considera como aceptable si:<br>● El documento deja claro que la prueba que se ejecutó y cuál fue el resultado obtenido.<br>● La documentación se encuentra bien organizada, es fácil de leer y su formato corresponde con los ítems indicados del documento 302.<br>● La documentación no debe superar las 15 páginas. | |

**Información importante**

Lectura de obligatorio: 07-09-2020 Plazo máximo de entrega: 15-10-2020
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 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. **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.