STATUS: OK	CLASE 1: Backend Arquitecture
ASPECTO	**DESARROLLO**
Habilidades y competencias	*1.- Planeacion y definicion de la arquitectura de código*
Instrucciones *(¿Qué vamos a hacer en este hito? En lo posible, si bien puede ser paso a paso, mantener lo más a alto nivel posible)*	*1.- Crear y planificar el DER del proyecto a implementar tomando en cuenta la siguientes consideraciones* *La aplicación consistirá en un e-commerce en el cual* - *Un Usuario podrá registrarse e ingresar a la aplicación mediante usuario y contraseña.* - *El Usuario registrado puede realizar compras de productos mediante un carrito de compras (solo una unidad de cada producto) emitiendo una Orden de compra que registra la información en un Detalle de Compras* - *Las Órdenes de compras son asociadas al Usuario y estas a su vez tienen asociado un Detalle de Compra con la información de los productos adquiridos* - *Un Usuario Administrador, tendrá la posibilidad de actualizar la información de los productos cargados en la base de datos así como actualizar stock o agregar imágenes mediante un servicio de nube.*
Tips	
Requisitos *(¿Qué esperamos que funcione o suceda para considerar la actividad como completa?)*	*Al finalizar este hito el alumno deberá tener la estructura básica del proyecto individual de e-commerce y una idea teórica de las entidades de la base de datos asi como sus relaciones.*

STATUS: OK	CLASE 2: Nest JS Fundamentals
ASPECTO	**DESARROLLO**
Habilidades y competencias	*1.- Inicialización de Proyectos en Nest* *2.- Configuracion basica de un proyecto*
Instrucciones *(¿Qué vamos a hacer en este hito? En lo*	*1.- Crear un proyecto en Nest JS bajo el nombre ecommerce-<usuario de github>* *2.- Crear los módulos Products, Users y Auth*

posible, si bien puede ser paso a paso, mantener lo más a alto nivel posible)	*3.- Crear sus respectivos controllers y services* *4.- Crear los endpoints GET /products, GET /users y Get /auth* *5.- Crear un middleware global que loguee la ruta, método y la fecha-hora en que se llamó al endpoint*
Tips	*Utiliza Nest CLI para inicializar el proyecto* *Recuerda "modularizar" el código para trabajar de forma ordenada*
Requisitos *(¿Qué esperamos que funcione o suceda para considerar la actividad como completa?)*	*Al finalizar este hito el alumno deberá tener un proyecto de node con la estructura básica del proyecto individual de e-commerce.* *Los endpoints principales del proyecto deberán ser capaces de recibir solicitudes desde el cliente y activar un middleware que nos permita identificar mediante un log en la terminal la ruta invocada.*

STATUS: OK	**CLASE 3: Nest JS Fundamentals II**
ASPECTO	**DESARROLLO**
Habilidades y competencias	*1.- Creación y uso de repositorios de Nest JS*
Instrucciones *(¿Qué vamos a hacer en este hito? En lo posible, si bien puede ser paso a paso, mantener lo más a alto nivel posible)*	*1.- Crear los repositorios para Users y Products.* *2.- Guardar las entidades en un array en memoria.* *3.- Cargar algunas entidades de prueba hardcodeadas, con las siguientes propiedades* ***Users*** *id:number* *email: string* *name: string* *password: string* *address: string* *phone: string* *country?: string | undefined* *city?: string | undefined* ***Products*** *id:number* *name: string* *description: string*

	price: number *stock: boolean* *imgUrl: string* *5.- Modificar los endpoints GET /products y GET /users para que devuelvan el array de entidades*
Tips	*Recuerda que los controllers sólo pueden comunicarse con el repositorio a través de los servicios.* *No olvides actualizar el array de providers*
Requisitos *(¿Qué esperamos que funcione o suceda para considerar la actividad como completa?)*	*Al finalizar este hito, el alumno deberá implementar de manera satisfactoria un repositorio para cada entidad del proyecto* *Los endpoints de la aplicación deben funcionar de manera correcta y devolver la información provista por cada repositorio*

STATUS: OK	**CLASE 4: Nest JS Routing**
ASPECTO	**DESARROLLO**
Habilidades y competencias	*1.- Creación de rutas para el procesamiento de solicitudes correspondientes al CRUD de cada entidad* *2.- Validación del cuerpo de las solicitudes* *3.- Implementar la paginación de resultados de acuerdo a la información recibida en la ruta* *4.- Uso de guardias para la protección de rutas*
Instrucciones *(¿Qué vamos a hacer en este hito? En lo posible, si bien puede ser paso a paso, mantener lo más a alto nivel posible)*	*1.- Crear todos los endpoints CRUD para Products y Users (GET, GET{id}, POST, PUT{id}, DELETE{id})* *2.- Desarrollar la lógica de creación, listado y eliminación desde el repository* ● *GET debe devolver la lista de elementos , y httpStatus = 200* ● *GET{id} debe devolver el elemento con id pedido, y httpStatus = 200* ● *En el caso de Users, no devolver el password* ● *POST debe devolver el id de la entidad creada, y httpStatus = 201* ● *PUT{id} y DELETE{id} pueden devolver el id de la entidad editada/eliminada y httpStatus 200.* *3.- Validar en POST y PUT que la estructura de la entidad corresponda a la estructura de cada entidad*

	4.- El método GET puede recibir como query params los valores page y limit ● *Si no recibe el parámetro page, el valor por defecto es 1* ● *Si no recibe el parámetro limit, el valor por defecto es 5* ● *Bonus: Implementar la lógica desde el repositorio, para paginar las entradas devuelvas* ● *5.- Crear el endpoint POST /auth/signin, que reciba email y password* ● *Para el login se utilizarán las credenciales email / password* ● *Inyectar el usersRepository para poder hacer consultas* ● *No se procederá al login si faltan alguna de las dos credenciales.* ● *No se procederá con el login en caso de que no exista un usuario registrado con la dirección de email proporcionada.* ● *En caso de que el usuario no exista o la contraseña proporcionada no coincida con la registrada, se deberá enviar una única respuesta para cualquiera de los casos. Ej: "Email o password incorrectos". NOTA: Por seguridad es preferible no especificar cuál de los dos datos ha fallado en su verificación.* *6.- Dentro de la carpeta Auth, crear una guarda AuthGuard, que debe verificar lo siguiente:* ● *Debe existir un header Authorization* ● *Dicho header, tiene que tener una estructura como la siguiente: Basic: <email>:<password>* ● ***NO validaremos por ahora que sea un email y un password válido, únicamente verificar si el header es enviado y continente un email y un password*** ● *Todos los endpoints de Users, salvo el POST, deben utilizar esta guarda* ● *Todos los endpoints de Products, salvo el GET y el GET{id} deben utilizar esta guarda*
Tips	*No te preocupes por ahora por el manejo de errores, la ruta de autenticación puede devolver strings únicamente.*
Requisitos *(¿Qué esperamos que funcione o suceda para considerar la actividad como completa?)*	*Al finalizar este hito el proyecto debe contar con una ruta para cada una de las acciones correspondientes al CRUD de cada entidad,* *La lógica de estas tareas deberá estar encapsulada en el repositorio correspondiente.* *Los endpoints deberán ser validados para asegurar la integridad de la información recibida en la solicitud* *Las rutas deberán ser protegidas por una guarda*

STATUS: OK	**CLASE 5: Nest JS y TypeORM**

ASPECTO	DESARROLLO
Habilidades y competencias	*1.- Creación de una base de datos en Postgres* *2.- Configuración y conexión con TypeORM* *3.- Creación y definición de entidades*
Instrucciones *(¿Qué vamos a hacer en este hito? En lo posible, si bien puede ser paso a paso, mantener lo más a alto nivel posible)*	*1.- Instalar y configurar las librerías necesarias para utilizar typeorm y postgres* ● *Crear un archivo de configuración para la conexión con typeorm* ● *Los datos de conexión a la BD deben ser almacenados en variables de entorno* ● *Crear el módulo de conexión de manera global* *2.- Definir las siguientes entidades de typeorm con sus respectivas relaciones* ***Users*** - ***id**: Debe ser un valor único generado automáticamente en formato UUID. No puede ser nulo y actúa como la clave primaria de la entidad.* - ***name**: Debe ser una cadena de texto de máximo 50 caracteres y no puede ser nulo.* - ***email**: Debe ser una cadena de texto de máximo 50 caracteres, único y no puede ser nulo.* - ***password**: Debe ser una cadena de texto de máximo 20 caracteres y no puede ser nulo.* - ***phone**: Debe ser un número entero.* - ***country**: Debe ser una cadena de texto de máximo 50 caracteres.* - ***address**: Debe ser un texto.* - ***city:** Debe ser una cadena de texto de máximo 50 caracteres.* - ***orders_id:** Relación 1:N con orders* ***Products*** - ***id**: Debe ser un valor único generado automáticamente en formato UUID. No puede ser nulo y actúa como la clave primaria de la entidad.* - ***name**: Debe ser una cadena de texto de máximo 50 caracteres y no puede ser nulo.* - ***description**: Debe ser un texto y no puede ser nulo.* - ***price**: Debe ser un número decimal con una precisión de 10 dígitos y una escala de 2 dígitos. No puede ser nulo.* - ***stock**: Debe ser un valor numérico. No puede ser nulo* - ***imgUrl**: Debe ser una cadena de texto, en caso de no recibir un valor debe asignar una imagen por defecto.* - ***category_id** (Relación 1:N)* - *Relación N:N con orderDetails* ***Categories***

- ***id**: Debe ser un valor único generado automáticamente en formato UUID. No puede ser nulo y actúa como la clave primaria de la entidad.*
- ***name**: Debe ser una cadena de texto de máximo 50 caracteres y no puede ser nulo.*
- *Relación 1:N con products*

Orders

- ***id**: Debe ser un valor único generado automáticamente en formato UUID. No puede ser nulo y actúa como la clave primaria de la entidad.*
- ***user_id:** (Relación 1:N) con users*
- ***date***
- *Relación 1:1 con orderDetails*

OrderDetails

- ***id:** Debe ser un valor único generado automáticamente en formato UUID. No puede ser nulo y actúa como la clave primaria de la entidad.*
- ***price:** Debe ser un número decimal con una precisión de 10 dígitos y una escala de 2 dígitos. No puede ser nulo.*
- *order_id: Relación 1:1 con orders*
- *Relación N:N con products*

3.- Al inicializar el servidor deben pre-cargarse a la base de datos las categorías y los productos del siguiente archivo a la base de datos. Ten en cuenta que la categoría debe ser cargada antes del producto.

- *Vamos a suponer que del lado del cliente se ejecutará el endpoint **/categories/seeder** para realizar la carga de categorías al inicializar la aplicación así que puedes contemplar este proceso como parte del flujo de un controlador normal. Lo mismo para la pre-carga de productos utilizando del endpoint /**products/seeder** (El orden de invocación es importante)*

- *Debes evitar la carga de registros bajo el mismo nombre tanto en categorías como en productos.*

- *Deberás crear el módulo, controlador, servicio y repositorio correspondiente para las categorías, este repositorio solo debe contener los métodos **getCategories** y **addCategories***

JavaScript

```
[
  {
    "name": "Iphone 15",
    "description": "The best smartphone in the world",
    "price": 199.99,
```

```
    "stock": 12,
    "category": "smartphone"
  },
  {
    "name": "Samsung Galaxy S23",
    "description": "The best smartphone in the world",
    "price": 150.0,
    "stock": 12,
    "category": "smartphone"
  },
  {
    "name": "Motorola Edge 40",
    "description": "The best smartphone in the world",
    "price": 179.89,
    "stock": 12,
    "category": "smartphone"
  },
  {
    "name": "Samsung Odyssey G9",
    "description": "The best monitor in the world",
    "price": 299.99,
    "stock": 12,
    "category": "monitor"
  },
  {
    "name": "LG UltraGear",
    "description": "The best monitor in the world",
    "price": 199.99,
    "stock": 12,
    "category": "monitor"
  },
  {
    "name": "Acer Predator",
    "description": "The best monitor in the world",
    "price": 150.0,
    "stock": 12,
    "category": "monitor"
  },
  {
    "name": "Razer BlackWidow V3",
    "description": "The best keyboard in the world",
    "price": 99.99,
    "stock": 12,
    "category": "keyboard"
  },
  {
    "name": "Corsair K70",
    "description": "The best keyboard in the world",
    "price": 79.99,
    "stock": 12,
```

```
    "category": "keyboard"
  },
  {
    "name": "Logitech G Pro",
    "description": "The best keyboard in the world",
    "price": 59.99,
    "stock": 12,
    "category": "keyboard"
  },
  {
    "name": "Razer Viper",
    "description": "The best mouse in the world",
    "price": 49.99,
    "stock": 12,
    "category": "mouse"
  },
  {
    "name": "Logitech G502 Pro",
    "description": "The best mouse in the world",
    "price": 39.99,
    "stock": 12,
    "category": "mouse"
  },
  {
    "name": "SteelSeries Rival 3",
    "description": "The best mouse in the world",
    "price": 29.99,
    "stock": 12,
    "category": "mouse"
  }
]
```

4.- Crear el modelo,controllador, servicio y repositorio para las órdenes de compra (orders), dentro de este repositorio crearemos la lógica necesaria para que un usuario pueda realizar una compra de un "carrito de productos".

- *La orden de compra será recibida mediante una solicitud de HTTP Post al endpoint /orders cuyo cuerpo tendrá la siguiente estructura:*

JavaScript

```
{
 "userId":"UUID del usuario",
 "products":[
  {
 "id":"UUID producto 1"
 },
 {
```

	 `` "id":"UUID producto 2" }] } ``` ● *Por ahora los usuarios solo pueden agregar una unidad de cada producto dentro de su carrito.* ● *En el repositorio de orders tendrás que crear 2 métodos diferentes getOrder y addOrder* ***addOrder*** - *Busca a un usuario por id* - *Crea un registro en la tabla orders con el usuario encontrado* - *Busca los productos por id recibidos en la request actualizando el total de la compra y reduciendo el stock del producto correspondiente. (al realizar la búsqueda de todos los productos aquellos con stock igual a 0 no deben ser mostrados)* - *Construye y registra un detalle de compra con los productos seleccionados* - *Devuelve la orden de compra con el precio y id del detalle de compra* ***getOrder*** - *Busca una orden recibida por id* - *Devuelve un objeto con la orden y los detalles de la orden (el detalle de la orden debe contener un array con todos los productos adquiridos)* *5.- Modificar los el contenido del repositorio para que utilice la entidad Users para la gestión de información. En el caso de la búsqueda por Id la respuesta debe devolver al usuario incluyendo un array con las órdenes de compras efectuadas (únicamente id y date)* *6.- Configurar las migraciones correspondientes y los comandos en el package.json para ejecutarlas*
Tips	*Tendrás que hacer algunos cambios en controladores y servicios para que funcione correctamente la aplicación en conjunto con la DB* *Puedes almacenar el archivo de productos en un json dentro de la carpeta src de tu proyecto*
Requisitos *(¿Qué esperamos que funcione o*	*Al terminar el hito el alumno debe haber realizado la correcta configuración de la base de datos en el proyecto.* *Los servicios deben trabajar con los repositorios de cada entidad para gestionar la*

suceda para considerar la actividad como completa?)	*información en la base de datos.* *Las relaciones entre tablas deben funcionar correctamente al realizar el proceso de compra* *Debe estar configurada la implementación de migraciones para monitorear futuros cambios en la base de datos.*

STATUS: OK	**CLASE 6: Nest JS Pipes**
ASPECTO	**DESARROLLO**
Habilidades y competencias	*Creación de validaciones por medio de Pipes para validar la la información de las solicitudes.*
Instrucciones *(¿Qué vamos a hacer en este hito? En lo posible, si bien puede ser paso a paso, mantener lo más a alto nivel posible)*	*1.- Implementar el global pipe de Class-Validator* *2.- Crear los DTOs CreateUserDto y CreateOrderDto, e implementarlos en los POST y PUT correspondientes* ***CreateUserDto*** - ***name**: Se requiere que el nombre no esté vacío, sea una cadena de al menos 3 caracteres y no supere los 80 caracteres de longitud.* - ***email**: El correo electrónico debe tener una estructura válida según el estándar de direcciones de correo electrónico.* - ***password**: La contraseña debe cumplir con los siguientes criterios:* ● *Debe contener al menos una letra minúscula, una letra mayúscula, un número y uno de los siguientes caracteres especiales: !@#$%^&** ● *Debe tener una longitud mínima de 8 caracteres y una longitud máxima de 15 caracteres.* - ***address**: La dirección debe tener una longitud mínima de 3 caracteres y no superar los 80 caracteres de longitud.* - ***phone:** El número de teléfono debe estar presente y ser un número.* - ***country:** El país debe ser una cadena de texto de al menos 5 caracteres y no superar los 20 caracteres de longitud.* - ***city**: La ciudad debe ser una cadena de texto de al menos 5 caracteres y no superar los 20 caracteres de longitud.* ***CreateOrderDto***

	• ***userId**: Se requiere que el userId no esté vacío y cumpla con el formato de UUID.* • ***products**: Se espera que products sea un array que contenga al menos un elemento. Cada elemento del array debe ser un objeto parcial de la entidad Products.* *3.- Crear el dto LoginUserDto, e implementarlo en POST /auth/signin* ***LoginUserDto*** • *Validar que el email tenga una estructura válida* • *Validar que el password contenga:* ○ *al menos una minúscula* ○ *al menos una mayúscula* ○ *al menos un caracter numérico* ○ *un largo mínimo de 8 caracteres* ○ *un largo máximo de 15 caracteres* ○ *al menos uno de los siguientes caracteres = !@#$%^&** • *Validar para el resto de campos, que se condigan con el dato de la BD, y que los campos string no superen el largo definido en la entidad* *4.- Extra: Agrega el manejo de errores que creas correspondiente para cada ruta* *5.- Validar en todos los endpoints que lo requiera, que el id tenga el formato especificado (UUID) recibido por parámetros o query.*
Tips	
Requisitos *(¿Qué esperamos que funcione o suceda para considerar la actividad como completa?)*	*Al terminar el hito el alumno debe haber implementado correctamente la validación de solicitudes HTTP por medio de Pipes* *Las validaciones deben ser implementadas en aquellos endpoints que utilicen información proveniente de la solicitud según corresponda*

STATUS: OK	**CLASE 7: File Upload**
ASPECTO	**DESARROLLO**
Habilidades y competencias	*1.- Conexión con Cloudinary para la gestión de imágenes en la nube* *2.- Subida de archivos a una sesión de Cloudinary* *3.- Consumo de imágenes de Cloudinary*
Instrucciones	*1.- Configurar una cuenta en Cloudinary y generar las credenciales de acceso correspondientes*

(¿Qué vamos a hacer en este hito? En lo posible, si bien puede ser paso a paso, mantener lo más a alto nivel posible)	*2.- Crear la conexión dentro de la aplicación mediante un archivo de configuración* *3.- Crear el módulo, servicio, controlador y repositorio correspondientes para la gestión de archivos.* *4.- Desarrollar la lógica para la carga de imágenes a Cloudinary y la actualización de imágenes de los productos en la DB. Este proceso será realizado por medio del endpoint* ***/files/uploadImage/:id*** *que recibe por parámetros el id del producto cuya imagen queremos actualizar y el archivo a emplear en el cuerpo de la solicitud.* *5.- La DB debe reflejar los cambios efectuados en el campo imgUrl* *6.- Implementar pipes para la validación del tamaño de imagen (no mayor a 200kb) así como los tipos de imagen permitidos.*
Tips	
Requisitos *(¿Qué esperamos que funcione o suceda para considerar la actividad como completa?)*	*Al terminar este hito el alumno debe haber implementado la actualización de productos para utilizar la carga de imágenes al servicio de cloudinary.* *Las imágenes deben tener una validación de tamaño y tipo antes de ser cargadas en el servicio de Cloudinary*

STATUS: OK	**CLASE 8: Autenticación**
ASPECTO	**DESARROLLO**
Habilidades y competencias	*1.- Implementación de la encriptacion y desencriptacion de contraseñas* *2.- Firma y validación de JWT* *3.- Uso de guardianes para la autenticación y protección de rutas*
Instrucciones *(¿Qué vamos a hacer en este hito? En lo posible, si bien puede ser paso a paso, mantener lo más a alto nivel posible)*	***Sign Up*** - *Sustituir el endpoint* ***POST /users*** *por el endpoint* ***POST /auth/signup*** *que será creado dentro del controlador de autenticación.* • *Este endpoint recibirá la misma estructura que recibia el endpoint anterior y adicionalmente recibirá una propiedad de confirmación de contraseña, debes validar que ambas contraseñas sean recibidas y coincidan o devolver una excepción.* • *Debe registrar al usuario dentro de la base de datos con una contraseña hasheada*

	● *Debe retornar al usuario sin contraseña* ***<u>Sign In</u>*** - *Modificar la funcionalidad de signIn para que valide el password encriptado con el provisto en la solicitud.* ● *Enviar un error genérico en caso de existir algún error ya sea por que el usuario no es encontrado o por que el password es incorrecto* ● *Crear un token de acceso para el usuario registrado con una validez de 1 hora* ***<u>Auth Guard</u>*** - *Modificar la funcionalidad del guardián de autenticación para la validación de tokens.* ● *Enviar un error en caso de no recibir el token o en caso de que este o sea un token válido con código de error 401* ● *El token debe ser verificado por medio de una clave secreta que no debe ser mostrada directamente en el código (Variables de entorno).* ● *Una vez validado el token debes adjuntar la información correspondiente al tiempo de expiración de dicho token* - *Los endpoints protegidos por este guardián serán los siguientes* ● *POST /uploadImage/:productId* ● *POST /orders* ● *GET /orders/:id* ● *PUT /products/:id* ● *GET /users* ● *GET /users/:id* ● *PUT /users/:id* ● *DELETE /users/:id*
Tips	*Recuerda modificar el DTO para la creación de usuarios* *Puedes utilizar decoradores personalizados para la validación.*
Requisitos *(¿Qué esperamos que funcione o suceda para considerar la actividad como completa?)*	*Al finalizar el alumno tendrá que haber implementado un sistema de autenticación por medio de la encriptación de contraseñas y la validación por medio de la gestión de tokens de JWT* *El proyecto deberá contar con rutas protegidas particulares y rutas públicas accesibles sin la necesidad de un token.*

STATUS: OK	CLASE 9: AUTENTICACIÓN II
ASPECTO	**DESARROLLO**
Habilidades y competencias	*Implementación del Control de acceso basado en roles*
Instrucciones *(¿Qué vamos a hacer en este hito? En lo posible, si bien puede ser paso a paso, mantener lo más a alto nivel posible)*	- *Definir un guardián para la validación del rol de administrador en usuarios para el control de acceso.* - *Modificar Entidad y Dtos para implementar el campo de administrador:* ● *Todos los registros serán considerados usuarios por default.* ● *El campo admin no debe ser recibido dentro de la solicitud.* ● *El campo admin no debe ser mostrado en las rutas que devuelven un usuario (únicamente en la ruta GET /users/)* - *Definir los roles de la aplicación (únicamente es necesario el rol de administrador).* - *Asignar y verificar de rol junto con el proceso de firma de JWT* - *Implementar de control de Acceso en las rutas:* ● *GET /users/* ● *PUT /products/:id*
Tips	*No olvides utilizar un custom decorator para la definición de roles*
Requisitos *(¿Qué esperamos que funcione o suceda para considerar la actividad como completa?)*	*Al terminar este hito la aplicación deberá contar con rutas protegidas por medio del Control de acceso basado en roles.*

STATUS: OK	CLASE 10: Testing
ASPECTO	**DESARROLLO**
Habilidades y competencias	*Implementación de pruebas unitarias y/o de integración*
Instrucciones	- *Crear e implementar pruebas unitarias en los diferentes módulos de la aplicación*

(¿Qué vamos a hacer en este hito? En lo posible, si bien puede ser paso a paso, mantener lo más a alto nivel posible)	- *Crear e implementar pruebas de integración en la aplicación*
Tips	*La implementación de pruebas es un extra credit para el proyecto integrador*
Requisitos *(¿Qué esperamos que funcione o suceda para considerar la actividad como completa?)*	*Para tomar el extra credit de este hito el alumno deberá haber creado:* - *5 pruebas unitarias de al menos 5 funcionalidades diferentes dentro de la aplicación.* - *Validar mediante pruebas de integración el funcionamiento de al menos 5 rutas de la aplicación.*

STATUS: OK	**CLASE 11: OPEN API**
ASPECTO	**DESARROLLO**
Habilidades y competencias	*Crear la documentación correspondiente para la aplicación utilizando swagger y Open API*
Instrucciones *(¿Qué vamos a hacer en este hito? En lo posible, si bien puede ser paso a paso, mantener lo más a alto nivel posible)*	- *Integrar swagger a la aplicación para la generación dinámica de la documentación en la ruta /api* - *Cada controlador debe tener su propia etiqueta para facilitar la legibilidad.* - *Mantener la protección de rutas que utilicen JWT, las rutas con control de acceso mediante roles únicamente pueden ser testeadas para validar errores.* - *Los DTOs y entidades deben estar detallados en la documentación* - *Las pruebas de la aplicación de forma integral en la interfaz de Open API deben ser funcionales*
Tips	*Puedes utilizar el formato con comentarios o decoradores para la definición y personalización de los DTOs y entidades*
Requisitos *(¿Qué esperamos que funcione o suceda para*	*Al finalizar el hito la aplicación debe mostrar la documentación COMPLETA de la aplicacion donde se desglosen las rutas, DTOs y entidades disponibles para el correcto uso de la API*

considerar la actividad como completa?)	

STATUS: OK	**CLASE 12: Docker**
ASPECTO	**DESARROLLO**
Habilidades y competencias	*Creacion de imagenes y componentes de Docker* *Uso de docker compose para el uso local de una aplicación*
Instrucciones *(¿Qué vamos a hacer en este hito? En lo posible, si bien puede ser paso a paso, mantener lo más a alto nivel posible)*	- *Definir la imagen de la aplicación dentro de un archivo docker* - *Crear un archivo docker-compose para utilizar la aplicación en conjunto con una base de datos dockerizada.* - *Asegurar la persistencia de datos del contenedor.*
Tips	
Requisitos *(¿Qué esperamos que funcione o suceda para considerar la actividad como completa?)*	*Al finalizar el hito el alumno deberá contar con una aplicación completamente montada en contenedores de docker que pueda ser levantada de forma integral mediante docker compose y preserve todas sus funcionalidades de manera local*