# GUÍA DE ESTRUCTURA DE APLICACIÓN BACKEND CON NODE JS BASADO EN ARQUITECTURA HEXAGONAL

1. Instalar nodejs: <https://nodejs.org/es/download/>

Descargar el ejecuta le y seguir los pasos de instalación.

Captura de pantalla de un video juego

Descripción generada automáticamente

1. Crear el package.json (archivo de meta descripción del proyecto): **npm init --yes** (la opción --yes es para que la creación se haga directa sin confirmaciones):

Texto

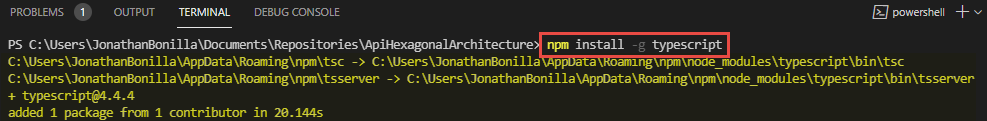
Descripción generada automáticamente

Texto

Descripción generada automáticamente

1. Instalar typescript: <https://www.typescriptlang.org/download>

Se abre una terminal integrada en visual studio code y se ejecuta el comando **npm install -g typescript** para instalar typescript de manera global mediante npm.



Texto

Descripción generada automáticamente

1. Crear archivo de configuración de typescript. Este archivo permite usar las directivas respectivas para la configuración de typescript en BackEnd (recordemos que también se puede usar en FrontEnd). Para crearlo, usamos el comando **tsc –init:**

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

**NOTA:** Si al momento de ejecutar el comando se presenta un error de permisos de ejecución de scripts:

Texto

Descripción generada automáticamente

Es necesario abrir la consola de PowerShell como administrador y levantar esa restricción con el siguiente comando **Set-ExecutionPolicy Unrestricted**:

Captura de pantalla de un celular

Descripción generada automáticamente

1. En el archivo **‘tsconfig.json’** creado, cambiamos en la opción **‘target’** por es6 (ECMAScript 6) ya que vamos a trabajar con async await para estructurar funciones asíncronas sin bloqueos y generar optimización de tiempos de respuesta:

Texto

Descripción generada automáticamente

1. Instalar módulo express (Framewrok de servidor para crear aplicaciones web y api en nodejs) con el comando **npm i express**:

Texto

Descripción generada automáticamente

1. Instalar módulo morgan (permite ver por consola las peticiones que van llegando, funciona como un middleware que registra la peticiones http) con el comando **npm i morgan**:

Texto

Descripción generada automáticamente

1. Instalar módulo helmet (que es realmente una colección de nueve funciones de middleware más paquetes que establecen cabeceras HTTP relacionadas con la seguridad para aplicaciones basadas en express para ambientes productivos) con el comando **npm i helmet**:

Texto

Descripción generada automáticamente

1. Instalar módulo cors (que permite la gestión del suministro y acceso de recursos o denegación de estos desde una aplicación cliente hacía la nuestra aplicación nodejs) con el comando **npm i cors**:

Texto

Descripción generada automáticamente

1. Instalar módulo compression (para comprimir el peso de las respuestas del BackEnd para al FrontEnd para aplicaciones basadas en express para ambientes productivos) con el comando **npm i compression**:

Texto

Descripción generada automáticamente

1. Los módulos instalados se alojan en la carpeta ‘node\_modules’ , adicional a esto se crea el archivo ‘package-lock.json’ que se encarga de presentar un snapshot del árbol de dependencias que estamos incluyendo en el proyecto. Los dependencias instaladas con sus diferentes versiones las podemos ver en el archivo ‘package.json’:

Texto

Descripción generada automáticamente

1. Ahora se debe crear la carpeta que contendrá todos los conceptos de negocio enmarcados en el estilo de Arquitectura Hexagonal:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

1. Dentro de la carpeta ‘concepts’ se debe crear el archivo de registro de express para construir la aplicación. Este se debe llamar ‘server.ts’:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

1. Ahora, debemos configurar los ‘Types’ de typescript que son complementos al lenguaje para que pueda interpretar los módulos instalados.

Comenzamos con el complemento de node que sirve para poder tener complementos cuando trabajemos con Typescript desde nodejs o para proyectos de BackEnd. El siguiente es el de express que sirve para poder trabajar y ejecutar express y a este la vamos a añadir dos módulos que nos ayudarán en el desarrollo en tiempo de ejecución. Realizamos todo esto con el comando de instalación **npm install @types/node @types/express @types/morgan @types/compressison @types/cors nodemon typescript -D** (la letra mayúscula ‘D’ es para que se instale como dependencia de desarrollo)**:**

Texto

Descripción generada automáticamente

Para visualizar que la instalación fue exitosa verificamos en el archivo ‘package.json’. Allí se c rea una sección nueva llamada ‘devDependencies’ que nos indica que estas dependencias no son necesarias para que el proyecto funcione (a diferencia de la sección ‘dependencies’ que si son obligatorias) pero que si nos ayudaran en el momento de desarrollar:

Texto

Descripción generada automáticamente

1. Para manejar un estándar de encabezados en todos los archivos es necesario instalar un complemento que nos permite insertar marca de tiempo, derechos de autor o cualquier información como por ejemplo comentarios:

<https://marketplace.visualstudio.com/items?itemName=doi.fileheadercomment>

La configuración de se hace mediante la edición del archivo json de la extensión:

Captura de pantalla de un celular

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Texto

Descripción generada automáticamente

1. Ahora empezamos a construir y codificar la instancia del servidor. En este archivo se define toda la lógica necesario para que el servidor arranque mediante express (ver el archivo en el repositorio de código para entrar en detalle de la lógica programada):

Captura de pantalla de computadora

Descripción generada automáticamente

1. Cuando se compila la aplicación, es necesario que nodejs transforme el código de Typescript a Javascript. Lo ideal es que está transformación de archivos quede separada de los archivos fuentes que se codifican en Typescript. Para que esto sea posible, es necesario habilitar la variable **‘outDir’** en el archivo **‘tsconfig,js’** y agregar un nombre de carpeta para que estos archivos transformados se alojen allí:

Captura de pantalla de computadora

Descripción generada automáticamente

1. Con el comando ‘tsc’ se realiza la compilación y transformación del código de Typescript a Javascript y este resultado se almacena en la carpeta definida:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

1. Para validar que la lógica programada y el servidor se encuentran funcionando, ejecutamos el comando **node build/server.js** (haciendo referencia al archivo transformado en la carpeta creada). En este caso en el inicio del servidor mostramos un mensaje por consola que indica porque puerto sale la aplicación (a modo de ejemplo):

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Desde el navegador:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

1. Con el fin de automatizar la transpilación de archivos en **ambiente de desarrollo** cuando se generan cambios en el código, debemos configurar en el archivo ‘package.json’ un script que permite convertir automáticamente el código de Typescript a Javascript y escucha los cambios que se presenten en el código (vigila los cambios en Typescript):

Captura de pantalla de computadora

Descripción generada automáticamente

Con el comando **npm run ts** activamos el proceso de compilación y vigilancia de cambios:

Texto

Descripción generada automáticamente

En este ejemplo se agrega un comentario al código y se evidencia que el servidor esta atento a esos cambios para poder compilarlos y generar la transformación de archivos automáticamente (de Typescript a Javascript):

Texto

Descripción generada automáticamente

1. También es posible automatizar la actualización de cambios a los archivos Javascript **en ambiente de desarrollo**, es decir, los archivos transformados que se guardan en la carpeta build. Para esto, debemos configurar en el archivo ‘package.json’ un script que vigila los cambios realizados al archivo ‘server.js’:

Captura de pantalla de un celular

Descripción generada automáticamente

Con el comando **npm run dev** ejecutamos el servidor que lee y vigila el archivo ‘server.js’:

Texto

Descripción generada automáticamente

1. Ahora, para automatizar la ejecución de transpilación y compilación del servidor en **ambiente productivo**, agregaremos otro comando (ojo, ejecutar solo en ambiente productivo). La diferencia aquí es que al ejecutar este comando nodejs ya no se queda vigilando los cambios:

Texto

Descripción generada automáticamente

1. Es necesario configurar las rutas a las que se accederá mediante el API. Para que esto quede independiente, se crea una carpeta llamada **‘routes’** y dentro de esta un archivo llamado **‘IndexRoutes.ts’** (ver el archivo en el repositorio de código para entrar en detalle de la lógica programada):

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

1. Es necesario evitar el acoplamiento directo entre las clases con el fin de poder generar un código mucho más mantenible, con evolución sencilla hacía los cambios que se puedan llegar a presentar y con una independencia total de comportamiento de los diferentes artefactos que generemos en el código. Para esto, es necesario aplicar al patrón de diseño orientado a objetos de **‘Inyección de Dependencias’.** Lo que busca este patrón es que en vez de generar un acoplamiento fuerte a través de la instancia de una clase dentro de otra, es que se inyecten todos los artefactos que la clase vaya a utilizar (dependencias). Ahora, para lograr esto, se va a implementar un IoC Container que básicamente es un contenedor de todas las dependencias que son necesarias para trabajar en una clase, las instancia y las proporciona a las clases respectivas. Para esto, vamos a instalar un paquete que ayuda a gestionar este contenedor de dependencias con el comando **npm i awilix**:

Texto

Descripción generada automáticamente

1. Para programar la lógica del contenedor se creará un archivo llamado **‘container.ts’** en la raíz del proyecto:

Texto

Descripción generada automáticamente