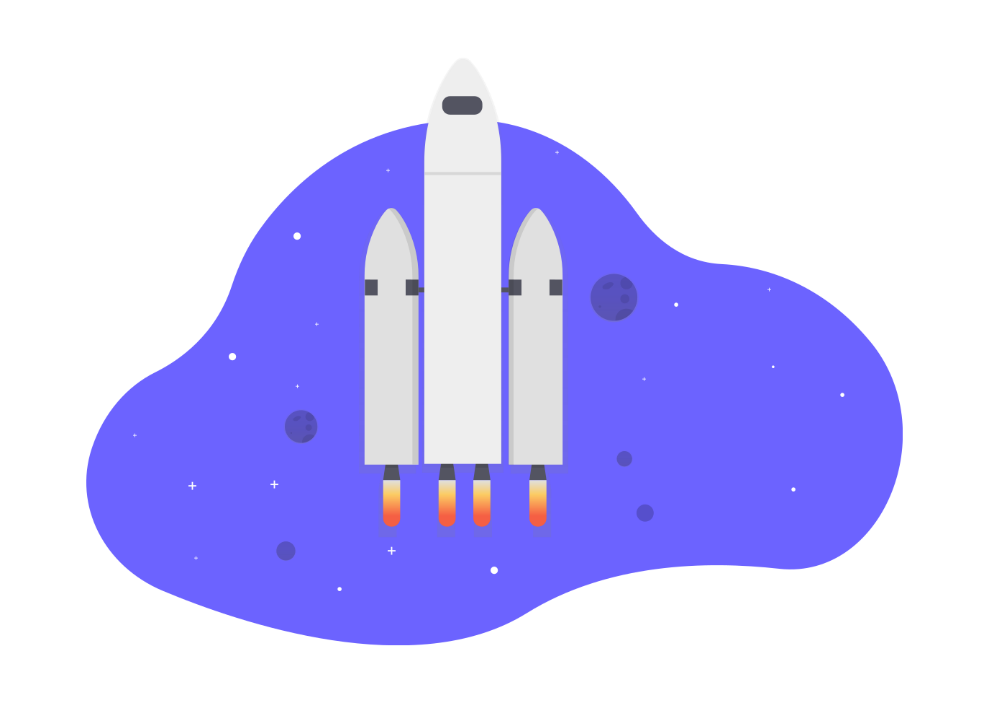
**Arquitectura de una aplicación**

*“Cualquier código tuyo que no hayas mirado en los últimos seis meses o más es como si lo hubiese escrito otro.” — Eagleson's Law*

Ya tienes el código totalmente funcional, las conexiones con la base de datos, los endpoints, que son cada punto de entrada de consulta de tu API, y un poco de seguridad incorporada. Prácticamente estás listo para despegar.



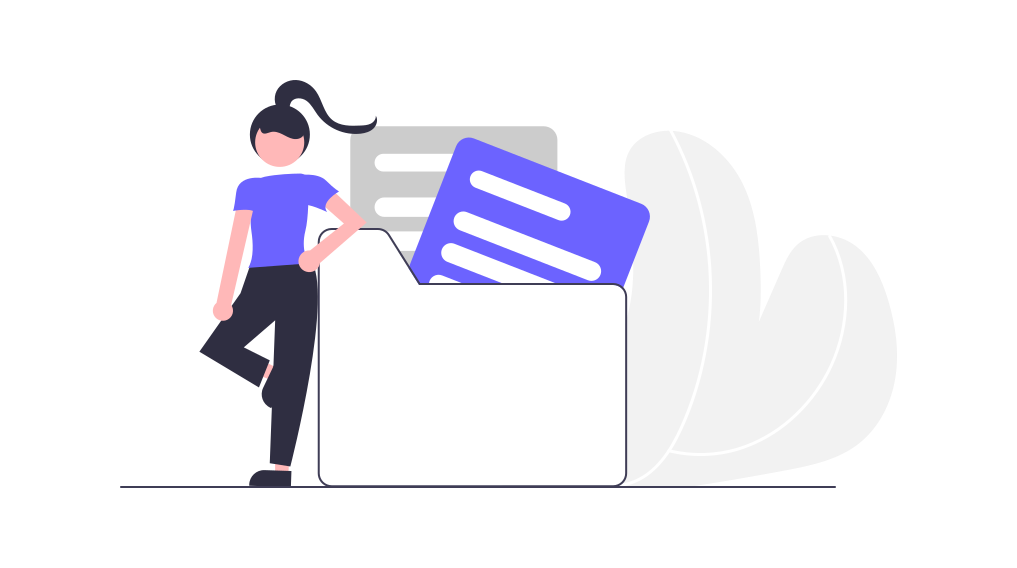
Pero antes, reflexiona un minuto: ¿te detuviste a pensar cómo y dónde guardaste el código?, ¿dónde está cada componente o archivo de tu aplicación? Si quisieras buscar dónde configuraste la conexión con tu base de datos, ¿te acordarías?

Seguramente al momento de hacerlo no habrás pensado en esto, ¡y es normal! Mientras el código funcione y haga lo que tiene que hacer, ¿por qué tenemos que preocuparnos por esto? Si bien no son requeridas para la construcción de una aplicación, estas y otras cuestiones te serán de mucha utilidad en algún momento de tu desarrollo. La arquitectura de una aplicación te proporcionará un plan y las prácticas recomendadas para obtener una aplicación bien estructurada. En esta toolbox veremos el motivo de su importancia.

**Mantenibilidad**

Cuando uno construye su aplicación, no suele pensar dónde crea un archivo o componente de su código ya que uno mismo sabe dónde está cada cosa, o al menos así es al momento de codear.

Ahora bien, ¿qué pasaría si han pasado meses desde la última vez que actualizaste tu código, o te toca actualizar el código de otra persona? La respuesta es simple: muchas veces hay que navegar todas las carpetas, una por una, y el código para saber de dónde viene cada llamado o archivo utilizado. ¿Suena tedioso, verdad?



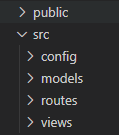
¡No se diga más! Por suerte existe gente que piensa en la [mantenibilidad](https://iso25000.com/index.php/normas-iso-25000/iso-25010/26-mantenibilidad)del código. La mantenibilidad refiere a qué tan fácil es mantener y actualizar el software en el tiempo. Si desde la concepción del código se tiene en cuenta esto, entonces al pensar en la mantenibilidad desde el comienzo, será más fácil de actualizar en el futuro. Es por esto que se crearon un conjunto de [buenas prácticas](https://es.wikipedia.org/wiki/Buenas_pr%C3%A1cticas) para que sea más fácil lograrlo.

Pero, ¿qué quiere decir esto? ¿Qué deberías hacer las cosas como te dicen que las haga? Realmente no, no es **obligatorio**hacerlas, pero te darás cuenta que ayudarán, y mucho. Incorporar estas buenas prácticas al momento de desarrollar habla de uno como *dev*,además de ayudarte y ayudar a los demás a actualizar código viejo. Recuerda siempre la importancia de la comunidad, como *dev*no estás solo en el mundo del desarrollo web, hay otrxs como tú. Ten siempre presente la red invisible que los une a todxs, y piensa en cómo ayudar a otrxs también implica ayudarte a tí mismo. Reflexiona: ¿cómo te sentirías si te toca trabajar con un código desarrollado por otro developer?, ¿preferirías que esté lo más organizado y claro posible, o que sea poco legible?

**¿Cómo funciona todo esto?**

La idea fundamental detrás de este concepto de mantenibilidad es estructurar la aplicación en carpetas de manera de poder juntar naranjas con naranjas y manzanas con manzanas. Por ejemplo, ¿qué pasaría si pones una pera el cajón del aguacate? Claramente, eso es factible, pero sería difícil encontrar, ¿verdad? Cuando programamos, sucede lo mismo. Te sugerimos siempre incluir lo que cumple el mismo rol en una misma carpeta, de manera que si tienes que buscar una vista o un [modelo](https://es.wikipedia.org/wiki/Modelo_de_datos), simplemente por el nombre de la carpeta podrás instintivamente saber dónde está cada una.

Veamos un ejemplo de esto:

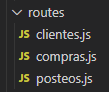


Separar las carpetas en categorías, como se demuestra en la imagen de arriba, permite, además de saber dónde está cada cosa, que tu código sea más prolijo y entendible. Para facilitar aún más este proceso, usualmente el/la desarrollador/a suele crear algunas carpetas dentro del directorio raíz del proyecto de trabajo para facilitar la organización, como pueden ser: ***src*** y ***public*** **.**

La carpeta ***src*** se suele identificar con aquella que contiene toda la lógica de la aplicación, mientras que la carpeta ***public*** **contiene todo lo ajeno y lo externo que puede ser visto por cualquier persona, por ejemplo el frontend de la aplicación. A su vez, esto permite añadir un extra de seguridad impidiendo que cualquier persona acceda a la carpeta***src*, **ya que dentro de la carpeta***src* se pueden tener los componentes, las vistas, las rutas, los modelos y las configuraciones que puedes llegar a usar en tu aplicación. Ten presente, que no necesariamente vas a usar siempre las mismas. Indaguemos un poco más en cada uno de estos conceptos para lograr tener a mano buenas prácticas que nos ayuden a organizar tanto nuestro código como a nosotrxs mismxs como developers.

**Buena Práctica #1: Routes**

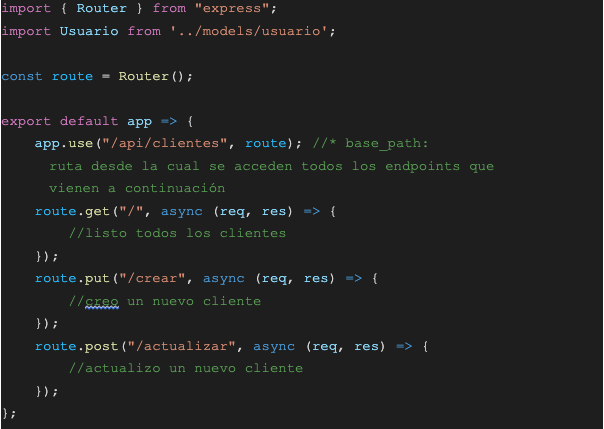
Además de separar en carpetas todos los endpoints que sirvan tu API, en vez de definirlos todos en un solo archivo, es una buena práctica separar los mismos en **rutas,**donde el *path*de la ruta coincida con la operación que queremos realizar.



Por ejemplo, si estás creando, actualizando y listando clientes en tu aplicación, puedes armar una ruta que sea:  “/clientes”, y dentro de ella incluir un endpoint para cada acción requerida:

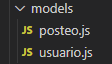
* “/clientes/crear”:
* “/clientes/actualizar”:
* “/clientes/”:

Como puedes observar, la misma raíz se puede usar para obtener todos los clientes. También puedes tener un archivo con el *base\_path*para cada ruta con sus propios endpoints.

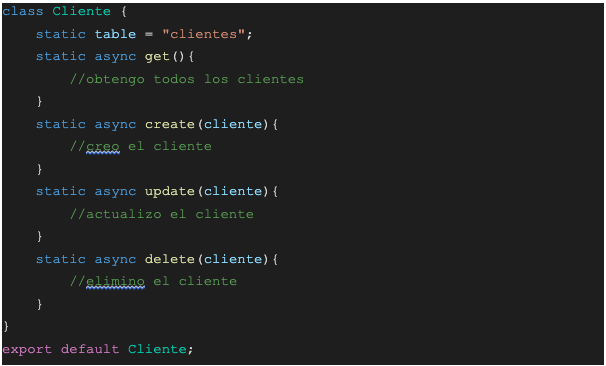


**Buena Práctica #2: Models**

Otra buena práctica es separar las entidades que tengamos en nuestro modelo en distintos archivos, de manera de separar, por ejemplo, ***Usuario***de ***Posteo,*** ya que los usuarios van a tener su relación con la tabla “usuarios” de la base de datos y los posteos la suya propia. Por ende, es más ordenado separarlos en distintos archivos para que sea más fácil su utilización y actualización con el tiempo.



Veamos este ejemplo de manera más concreta: tenemos nuestra tabla de “usuarios” en la base de datos, y nuestro modelo Usuario, donde definimos sus atributos y métodos para “crear”, “eliminar” y “actualizar”.



De esta manera, al separar cada modelo en un archivo distinto, es más fácil de buscar. También esto te permitirá no mezclar métodos al importar los modelos, y así poder usarlos en otras partes del código. Por el contrario, si tuvieras todos los modelos en un solo archivo, este puede hacerse muy largo y tal vez te confunda al querer actualizar el método “crear” del modelo ***Usuario***con el método “crear” del modelo ***Posteo*** **.**

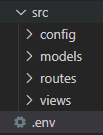
**Buena Práctica #3: Variables de entorno**

Hemos visto una primera aproximación a las variables de entorno al final del sprint anterior, las cuales serán de mucha ayuda si tienes varios entornos donde correrá tu aplicación, como pueden ser: entornos locales, de test o de producción. Seguramente en este punto en el que te encuentras, tampoco hayas pensado todavía dónde vas a alojar y correr tu aplicación. Por lo tanto ahora es un buen momento para empezar a hacerlo y tener en cuenta estas variables.

**¿Qué son las variables de entorno?**

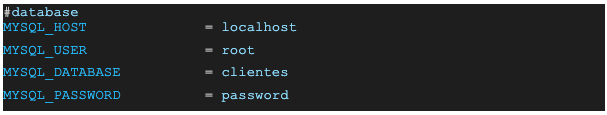
Las [variables de entorno](https://es.wikipedia.org/wiki/Variable_de_entorno) son campos que varían según la configuración que se les defina, lo que permite que según el entorno en el que se corra la aplicación, cargue un valor u otro. Por ejemplo, supongamos que tienes tu entorno de desarrollo local, con la conexión de base de datos también local y luego despliegas el código en un entorno productivo conectado a otra base de datos productiva. Entonces, tendrías que editar el código en cada entorno cambiando las conexiones a las base de datos correspondientes.

Haciendo uso de estas variables te evitarás cambiar el código en cada entorno. Sólo deberás configurar un archivo, sin preocuparte de tener que estar cambiando los valores en cada uno de los entornos. Esto ayudará y facilitará mucho el despliegue y mantenimiento del código en distintos entornos, además te permitirá cambiar dichas configuraciones sin tener que cambiar nada dentro de la aplicación.

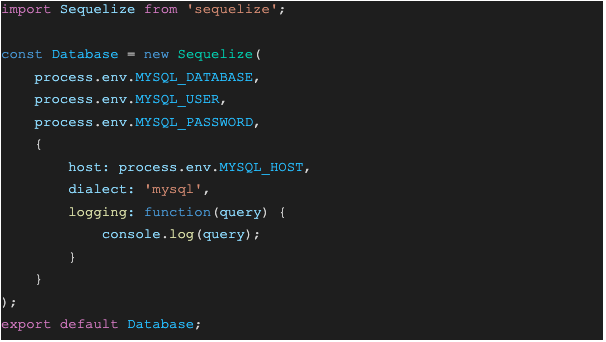


En aplicaciones de NodeJs es común crear un archivo llamado **“** ***.env”***en el raíz del proyecto. Aquí puedes definir todos los valores relacionados al entorno donde esté corriendo la aplicación. Como hemos visto, la aplicación puede correr en tu máquina, en algún servidor interno de la compañía para la que trabajes o en producción para que acceda todo el mundo.

Veamos cómo crear tu archivo  **“** ***.env”*** en el raíz.



Finalmente, en cualquier archivo de tu código fuente si haces uso de dichas variables, te devolverán el valor que tiene asignado en el archivo de configuración.



**Cierre**

De ahora en adelante, será importante que al momento de empezar tu sprint project, te detengas un momento a pensar todo lo que acabas de leer. El uso de estas buenas prácticas sin dudas te ayudarán a la hora de armar y mantener tu sprint project, no sólo para ti (la persona que lo está haciendo), sino también para la persona que lo vaya a actualizar (que puede ser otra persona que la que lo creó). Por eso, incorporarlas en tus hábitos de desarrollo es una buena forma de crecer, ya que no sólo estarás pensando en la solución, sino que al momento de construir tu aplicación o página web, tendrás en cuenta todos estos aspectos, que traducen el código más amigable, en simples palabras, en un mejor código