



Curso de Back-End con Node.js (Inicial)

Clase 09



Temario



Temario

- File Uploads.
- Deployment.
 - Vercel.
 - Supabase.
 - Amazon S3.



File Uploads

Subir y guardar archivos



File Uploads (1/4)

En una aplicación web es usual que los usuarios suban una foto de perfil, un video, un CV en PDF, etc.

Los archivos que se suben a un sitio se pueden guardar en:

- Filesystem (sistema de archivos del servidor).
- [Amazon S3](#).
- [Rackspace Cloud Files](#).
- [Google Cloud Storage](#).
- [Microsoft Azure Storage](#).
- Etc.



File Uploads (2/4) – Formulario

Para poder adjuntar un archivo en un formulario HTML simplemente se utiliza un `<input>` con `type="file"`. También se debe agregar `enctype="multipart/form-data"`:

```
<form action="..." method="post" enctype="multipart/form-data">
  ...
  <div class="mb-3">
    <label for="image" class="form-label">Seleccionar una imagen</label>
    <input id="image" name="image" type="file" class="form-control">
  </div>
  ...
</form>
```

Nota: este campo es poco personalizable en cuanto a su aspecto. Cada browser / sistema operativo lo muestra de forma diferente. Pero hay soluciones, por ejemplo: <https://getbootstrap.com/docs/5.1/forms/form-control/#file-input>.



File Uploads (3/4) – Formidable

Para agilizar el trabajo de procesar un archivo recibido, utilizaremos un módulo llamado [formidable](#). Para instalarlo, ejecutar el comando:

```
npm i formidable
```

Luego importarlo con:

```
const formidable = require("formidable");
```



File Uploads (4/4) – Ejemplo de uso

```
app.post("/articulos", (req, res) => {  
  
  const form = formidable({  
    multiples: true,  
    uploadDir: __dirname + "/public/img",  
    keepExtensions: true,  
  });  
  
  form.parse(req, (err, fields, files) => {  
    // Hacer algo con fields y files...  
    res.redirect("articulos");  
  });  
  
});
```

⚠ Consultar la documentación de Formidable para ver las distintas opciones de configuración.

⚠ Dentro del objeto `fields` quedan disponibles todos los campos “normales” del formulario, mientras que dentro del objeto `files` quedan disponibles todos los archivos que se subieron a través del formulario. Sugerencia: hacer `console.log` de `fields` y `files`.



Ejercicio 1

Ejercicio 1

- Crear un sitio web que conste de dos páginas:
 - Crear usuario (formulario).
 - Listado de usuarios.
- Al crear un usuario se debe ingresar:
 - Nombre.
 - Apellido.
 - Avatar (imagen de perfil).
- Las imágenes se deben guardar en la carpeta `/public/img`.

Ejercicio 1 (cont)



Ejemplo:

Crear usuario

Nombre

Apellido

Avatar

No file chosen

Listado de Usuarios

Id	Nombre	Apellido	Avatar
1	María	Pérez	
2	Juan	López	
3	Victoria	Gómez	
4	Pedro	Rodríguez	



Deployment



*“Terminamos de desarrollar nuestro proyecto,
¿ahora cómo lo publicamos?”*



Software Deployment (1/3)

Son todas las actividades que se deben llevar a cabo para **poner el software a disposición del usuario** (para que lo pueda usar).

Si se va poner el software a disposición del usuario final, se habla de “*Deploy to Production*”.

En las **aplicaciones web**, esto implica **colocar el software en un servidor** público o privado. En general se usan servidores públicos, pero a veces hay aplicaciones que sólo están disponibles en la red de una organización (ej: Intranet).



Software Deployment (2/3)

La tarea de configurar un servidor **suele ser algo complicado** y generalmente está en manos de un [SysAdmin](#) o un [DevOps](#); no en las de un desarrollador.

Por ejemplo, hay que tener en cuenta temas como:

- Instalación y configuración de un servidor (ej: Apache o Nginx).
- Instalación y configuración de una base de datos (ej: MySQL, MongoDB).
- Instalación de extensiones (como las que se requieren en PHP).
- Instalación y configuración de un firewall.
- Configuración de los permisos de los archivos.
- Gestión de backups.
- Instalación y configuración de un balanceador de cargas.
- Gestión de actualizaciones.
- Etc, etc, etc.

Pero si les interesa el tema, pueden ver tutorial: <https://serversforhackers.com/s/start-here>.



Software Deployment (3/3)

Una de las primeras tareas a realizar es **elegir un hosting** para la aplicación.

Se podría contratar una máquina virtual en [AWS EC2](#) o [DigitalOcean](#) pero, si bien son excelentes opciones, requieren de ciertos conocimientos que escapan a los objetivos de este curso.

Por suerte, existen servicios como [Heroku](#), [Vercel](#) y [AWS Elastic Beanstalk](#) que simplifican mucho la tarea de deployment. Son servicios que tienen como premisa que el **programador se enfoque en su código** y no la infraestructura.

Nota: Recientemente DigitalOcean lanzó un producto llamado [App Platform](#), que está en línea con Heroku y Vercel.



Deployment @ Vercel

Vercel (1/6)



Develop. Preview. Ship.

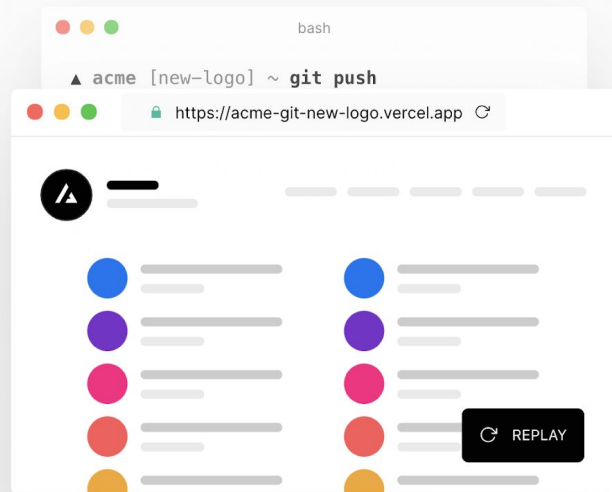
Vercel is the optimal workflow for frontend teams.

All-in-one: Static and Jamstack deployment,
Serverless Functions, and Global CDN.

Deploy Free

Contact Sales

Takes 15 seconds





Vercel (2/6)

[Vercel](#) (antes llamado Zeit) es un servicio de **hosting** que permite que el desarrollador se concentre en el desarrollo de las apps y no “perder tiempo” en tareas de SysAdmin o DevOps.

Algunas de sus características son:

- Está construido “arriba” de [AWS Lambda](#).
- Es **serverless** → el desarrollador no debe preocuparse de la infraestructura y sólo se paga por el tiempo real de uso.
- Tiene un **servicio gratuito excelente**, que puede ser usado en producción (no sólo para *tests*).

Vercel (3/6)

Vercel permite importar el código de una aplicación presente en un **repositorio Git**. Para ello será necesario crearse una cuenta en Vercel (crear un proyecto) y especificar la URL del repositorio Git que se utilizará. Además, Vercel permite **automatic deploys**, por lo que cada vez que haya una modificación en el repositorio, automáticamente se hará el *deploy* en Vercel.





Vercel (4/6)

Otras características de Vercel:

- 🎉 Incluye HTTPS por defecto.
- 🎉 Permite crear **URLs** como: <https://mi-app.vercel.app>.
- 🎉 Permite configurar **dominios propios** (ej: miempresa.com).
- 😞 No permite alojar archivos subidos por los usuarios en el *file system*.
- 😞 No incluye **base de datos** ni **cloud storage** (hay que conseguirlo a parte).
- Es necesario configurar las **variables de entorno** (que normalmente se colocan en un archivo `.env`) desde el *dashboard* de Vercel.



Vercel (5/6)

Si bien lo ideal suele ser linkear Vercel con nuestra cuenta de **GitHub** (y configurar *automatic deploys*), también se puede “deployar” la aplicación con un simple comando de consola:

```
vercel
```

Para hacer uso del mismo hay que previamente instalar [Vercel CLI](#) de forma global:

```
npm i -g vercel
```

Ver [detalles](#).



Vercel (6/6)

A la hora de “deployar” una aplicación **Node.js/Express** en Vercel, es necesario crear un archivo llamado **vercel.json** en la carpeta raíz de la aplicación:

```
{
  "version": 2,
  "builds": [
    { "src": "server.js", "use": "@vercel/node" },
    { "src": "/public/**/*", "use": "@vercel/static" }
  ],
  "routes": [
    {
      "src": "/css/(.*)",
      "dest": "/public/css/$1"
    },
    {
      "src": "/(.*)",
      "dest": "/server.js"
    }
  ]
}
```

Con ****/*** se indica: “cualquier archivo en cualquier sub-carpeta”. Ver detalles sobre este formato [aquí](#) y [aquí](#).

Esto es una expresión regular.

Con este JSON le indicamos a Vercel cómo debe ejecutar los archivos que componen la aplicación. Ver [detalles](#).



Supabase



Supabase (1/2)

Dado que Vercel no brinda almacenamiento para bases de datos, es necesario buscar otro proveedor.

😞 Lamentablemente no hemos encontrado un servicio (parcialmente) gratuito y fácil de usar para bases de datos **MySQL**.

🎉 Por eso sugerimos usar un servicio llamado [Supabase](#), que si bien no dispone de bases de datos MySQL, brinda un excelente servicio para bases de datos **PostgreSQL**. Si están usando Sequelize, dicho cambio será “transparente”.

Supabase brinda hasta **500 MB** de almacenamiento sin costo.





Supabase (2/2)

Luego de crearse una cuenta en Supabase, deberán crear un **proyecto** y elegir una contraseña para el mismo. Notar que este proceso podría demorar unos minutos.

Dentro del panel de control de Supabase encontrarán los datos necesarios para conectarse a la base de datos del proyecto. Ejemplo:

- DB Host: db.yudeelhwytrbiatzsnqg.supabase.co
- DB Port: 5432
- DB Name: postgres
- DB User: postgres
- DB Pass: (la que crearon para el proyecto)

Estos datos se deberán colocar en el archivo
`.env` (o en las variables de entorno de Vercel).



Supabase y Sequelize (1/2)

Si están usando Sequelize, hacer el cambio de MySQL a PostgreSQL es muy simple. Primero que nada deberán instalar las librerías `pg` y `pg-hstore`:

```
npm i pg pg-hstore
```

Luego se deberá cambiar la variable de entorno:

```
DB_CONNECTION=mysql
```

por:

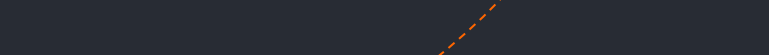
```
DB_CONNECTION=postgres
```



Supabase y Sequelize (2/2)

Si además usan Vercel, deberán indicarle (explícitamente) a Sequelize, que usarán la librería `pg`. Esto es necesario para evitar un problema que surge al hacer *deploy*.

```
const sequelize = new Sequelize(  
  process.env.DB_DATABASE,  
  process.env.DB_USERNAME,  
  process.env.DB_PASSWORD,  
  {  
    host: process.env.DB_HOST,  
    dialect: process.env.DB_CONNECTION,  
    dialectModule: require("pg"),  
    logging: false,  
  }  
);
```





Supabase – Cloud Storage



Supabase – Cloud Storage (1/4)

Además de alojamiento para bases de datos, Supabase también cuenta con un servicio de *cloud storage* para almacenar archivos (subidos por los usuarios).

Los archivos pueden ser imágenes, PDFs, ZIPs, etc. Incluso podrían ser archivos HTML, CSS y JavaScript.

Ver documentación: <https://supabase.io/docs/guides/storage>.





Supabase – Cloud Storage (2/4)

Instructivo para usar [cloud storage](#):

1. Ingresar al panel de control de Supabase e ingresar al **proyecto** con el que se esté trabajando.
2. Ingresar a la sección “**storage**” del proyecto.
3. Crear un “**bucket**” (público). → Un *bucket* es como una especie de carpeta donde se guardarán los archivos, a la cual se le pueden asignar diferentes tipos de permisos (políticas).
4. Crear una “**policy**” (política) para el *bucket* creado en el paso anterior, que permita crear archivos dentro del *bucket* (INSERT). Ver ejemplo en la siguiente diapositiva.



Supabase – Cloud Storage (3/4)

Ejemplo de una *policy*:

```
(  
  (bucket_id = 'avatars':: text)  
  AND (role() = 'service_role':: text)  
)
```

Esta *policy* establece que el usuario con rol “*service_role*” tendrá acceso al *bucket* llamado “*avatars*”.

Usando *policias* también se podría configurar para que sólo se puedan subir archivos de cierto tipo y/o en determinar sub-carpeta del *bucket*. Por más información, [entrar aquí](#).



Supabase – Cloud Storage (4/4)

5. Ir a la sección “**settings**” del proyecto, sub-sección “**API**” y conseguir los siguientes datos: URL y API Key (`service_role`).
6. Desde la aplicación de Node.js/Express, hacer una llamada a la API de Supabase para subir un archivo.

El último paso se puede llevar a cabo muy fácilmente gracias a una [librería JavaScript](#) creada por Supabase para justamente para poder interactuar con la plataforma de manera sencilla:

```
npm i @supabase/supabase-js
```

También es posible llamar a la API “a mano”, usando una librería como Axios o NodeFetch.



Ejercicio 2



Ejercicio 2

1. Crear un **repositorio** privado en **GitHub** y subir alguno de los ejercicios realizados durante el curso. Puede ser el último ejercicio realizado.
2. Crear archivo **vercel.json** en la raíz del proyecto.
3. Crearse una cuenta en **Supabase** (la cual servirá para guardar archivos así como información en una base de datos).
4. Crearse una cuenta en **Vercel**.
5. Crear un nuevo **proyecto en Vercel**.
 - a. Vincular el proyecto de Vercel con el repositorio de GitHub.
 - b. Crear variables de entorno en la sección *settings* del proyecto.



Amazon S3 – Cloud Storage



Amazon S3 (1/4)

En lugar de guardar los archivos subidos por los usuarios en el *file system*, es posible guardarlos en un servicio externo (como [Supabase Storage](#)), lo cual tiene varias ventajas.

[Amazon S3](#) es un servicio de [Amazon Web Services \(AWS\)](#) cuya única finalidad es la de guardar archivos en la “nube”. Los archivos pueden ser imágenes, PDFs, ZIPs, etc. Incluso podrían ser archivos HTML, CSS y JavaScript.

Sin embargo, Amazon S3 no sirve alojar para código de back-end que necesite ejecutarse. Para eso hay otros servicios de Amazon como EC2 o Lightsail.

Nota: Amazon ofrece gran parte de sus servicios (incluido S3) de forma gratuita durante 12 meses. Esto se conoce como [Amazon Free Tier](#). Es posible que deban ingresar una tarjeta de crédito (o prepaga) internacional. En estos casos a veces te debitan algunos centavos de dólar para verificar que la tarjeta es real.



Amazon S3 (2/4)

Instructivo para usar **Amazon S3**:

1. Crearse una **cuenta** en **AWS** (<https://aws.amazon.com>).
2. Crear un **Bucket** de **S3** (que es donde residirán los archivos) [aquí](#).
 - a. Elegir como región "us-east-1" (aunque podría ser otra).
 - b. Habilitar acceso público.
 - c. El resto de las opciones pueden quedar las de por defecto.
3. Crear un **User** de **AWS**, [aquí](#). No confundir este usuario con su “usuario personal”, creado en el paso #1. Al crear este usuario, obtendrán un **Access Key ID** y un **Secret Access Key**.



Amazon S3 (3/4)

4. Colocar todos datos obtenidos en el archivo `.env`. Ejemplo:

```
AWS_S3_BUCKET_NAME=ha-node  
AWS_S3_BUCKET_REGION="us-east-1"  
AWS_S3_API_VERSION="2012-10-17"  
AWS_USER_ACCESS_KEY_ID=AKIAXKUUEEMW3WP6P3MC  
AWS_USER_SECRET_ACCESS_KEY=nKKcAMhtDSP385u9arUEIZeENI1ogKPsObYFIRD
```

5. Darle **permisos** al usuario creado en el paso #3 para poder acceder al Bucket creado en el paso #2. Para esto deberán crear una **policy** para dicho usuario (ver siguiente diapositiva).
6. Instalar la librería [aws-sdk](#) en el proyecto, la cual simplifica el trabajo con S3.
7. Usar [este código como ejemplo](#) para conectarse a Amazon S3 y subir un archivo a su Bucket.



Amazon S3 (4/4)

Ejemplo de una *policy* (política) en S3. Es un archivo JSON.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3::mi-bucket",
        "arn:aws:s3::mi-bucket/*"
      ]
    }
  ]
}
```

Con esta *policy* se está diciendo que el usuario tiene acceso total (*) al *bucket* llamado *mi-bucket*.