

Sprint 2 – Ejercicios

Índice

Índice	1
Objetivo	2
Comentarios generales	2
Ejercicio 1	3
Ejercicio 2	3
Ejercicio 3	4
Ejercicio 4	5
Ejercicio 5	6
Ejercicio 6	7
Ejercicio 7	8
Ejercicio 8	9
Ejercicio 9	10
Ejercicio 10	11
Ejercicio 11	12
Ejercicio 12	13
Ejercicio 13	14
Ejercicio 14	15
Ejercicio 15	16
Ejercicio 16	18
Ejercicio 17	19
Ejercicio 18	19
Ejercicio 19	20
Ejercicio 20	21

Ejercicio 21	24
Ejercicio 22	26
Ejercicio 23	26

Objetivo

El objetivo de este Sprint es aprender sobre:

- Node.js.
- npm.
- Protocolo HTTP.
- Express.
- Middlewares.
- MySQL.
- MVC.

Comentarios generales

- Leer en detalle la pauta de cada ejercicio **antes de empezar a escribir código**.
- Notar que algunos ejercicios requieren que se haya dictado una clase previa (teórico) antes de poder resolverlos. Estos ejercicios estarán debidamente señalizados.
- En caso de **dudas**, pueden recurrir a sus compañeros, docentes (por Teams/Discord) y/o sitios en Internet (ej: Stack Overflow). Recuerden la importancia de apoyarse entre ustedes ya que una gran forma de aprender y reforzar conocimientos es explicarle a otro.
- También recomendamos tener una carpeta llamada **ha_bootcamp_sprint2** (o similar) para tener todos los ejercicios de este sprint juntos.

Ejercicio 1

Instalar [Node.js](#) y [npm](#) en tu equipo (si es que aún no lo has hecho).

Ejecutar el comando `node -v` para **verificar** que Node.js haya sido instalado correctamente. En la terminal debería aparecer el número de la **versión** de Node.js que se instaló.

⚠ Además, es muy aconsejable que hayas configurado tu sistema operativo para poder ver todos los **archivos y carpetas ocultas**, así como las extensiones de todos los archivos.

En Windows:

- [Mostrar archivos ocultos](#).
- [Mostrar todas las extensiones](#).

En Mac:

- [Mostrar archivos ocultos](#).
- [Mostrar todas las extensiones](#).

Ejercicio 2

Clase previa: “Node/npm”.

Pauta:

1. Crear una carpeta llamada `ejercicio_slugify`.
2. Inicializar un proyecto con el comando `npm init`.
Pueden ignorar todas las preguntas haciendo “Enter” en cada una.
3. Verificar que se haya creado el archivo `package.json`.
4. Instalar un paquete (dependencia) llamado [Slugify](#).
5. Verificar que se haya creado la carpeta `node_modules` y que se haya actualizado el archivo `package.json`.
6. Crear un archivo `index.js` dentro de la carpeta `ejercicio_slugify`.

7. Importar (requerir) en dicho archivo la dependencia:

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

8. Usar Slugify para convertir el texto:

```
"¡Quiero viajar a Bélgica & España! 🇧🇪🇪🇸"
```

a:

```
"quiero-viajar-a-belgica-y-espana".
```

9. Al llamar al archivo `index.js` (con el comando `node index.js`) deberá aparecer el nuevo texto impreso en la consola.

Ejercicio 3

Clase previa: "Node/npm".

Pauta:

1. Seguir trabajando con la carpeta del ejercicio anterior.
2. Importar el módulo HTTP en el archivo `index.js`.
3. Crear un servidor dentro del archivo `index.js` en el puerto 8080.
Se tendrá que usar la función `createServer`.
4. En el *callback* de la función `createServer`, colocar solamente esta línea de código: `console.log("Alguien accedió al servidor");`
5. En la terminal, ejecutar `node index.js`.
6. Abrir un navegador en <http://localhost:8080>. ¿Qué sucede en el navegador?
¿Qué sucede en la terminal? ¿Cómo se explica?
7. Editar el archivo `index.js`. Agregar dentro del *callback* de `createServer` la siguiente línea de código: `res.end("Respuesta");` (debajo del `console.log`). Guardar el archivo.
8. Antes de abrir el navegador, es necesario re-iniciar el proceso en la terminal. De lo contrario no se verán los cambios. Una forma es cerrando la terminal y volviéndola a abrir. Otra forma es escribir `CTRL+C`. Luego será necesario ejecutar nuevamente `node index.js`.
9. Ir a <http://localhost:8080>. ¿Qué sucede ahora en el navegador y en la terminal? ¿Tiene sentido la cantidad de mensajes en el terminal? Abrir la

pestaña Network de los Developer Tools. ¿Qué se puede ver? Recargar la página con F5 y CMD+R si es necesario.

Ejercicio 4

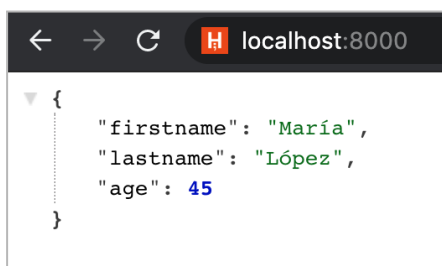
Clase previa: “Node/npm”.

Pauta:

1. Crear una carpeta llamada `ejercicio_json`.
2. Inicializar un proyecto con el comando `npm init`.
Pueden ignorar todas las preguntas haciendo “Enter” en cada una.
3. Verificar que se haya creado el archivo `package.json`.
4. Crear dos archivos: `index.js` y `persona.js`.
5. En `index.js` crear un servidor con `createServer`.
6. En `persona.js` crear un objeto JavaScript que contenga datos de una persona. Ejemplo:

```
{
  firstname: "María",
  lastname: "López",
  age: 45,
};
```

7. En `index.js` se debe importar (requerir) `persona.js`, y colocar su contenido en una variable llamada `persona`.
8. En el *callback* de `createServer` se debe escribir un código tal que cuando un navegante ingrese al sitio, aparezcan los datos de la persona en formato JSON.
9. El servidor deberá estar escuchando en el puerto 8000.
10. Ingresar a <http://localhost:8000> y ver el JSON en la pantalla. El navegador debería darse cuenta que recibió un objeto JSON. Si tienen la extensión JSON Formatter de Chrome deberían ver algo así:



⚠ Verificar que el navegador realmente reciba una respuesta en formato JSON, y no un texto plano.

Ejercicio 5

Clase previa: “Node/npm”.

Pauta:

1. Crear una carpeta llamada `ejercicio_rutas`.
2. Inicializar un proyecto con el comando `npm init`.
Pueden ignorar todas las preguntas haciendo “Enter” en cada una.
3. Verificar que se haya creado el archivo `package.json`.
4. Crear un archivo: `index.js`.
5. La idea es levantar un servidor en el puerto 8000, pero la idea ahora es hacer funcionar estas 3 URLs. Para cada URL se debe mostrar un texto diferente en la pantalla.
 - a. <http://localhost:8000> → “Home”.
 - b. <http://localhost:8000/productos> → “Productos”.
 - c. <http://localhost:8000/contacto> → “Contacto”.
6. Probablemente tengan que acceder al atributo `url` del objeto `request` (ver más [aquí](#)).
7. Mejorar la solución moviendo la lógica de rutas a un archivo separado llamado `routes.js`. El código debería funcionar igual que antes, pero ahora las responsabilidades deberían quedar mejor repartidas gracias a la modularización.

Ejercicio 6

Clase previa: “Node/npm”.

Con Node.js es posible acceder al disco duro de la máquina donde está corriendo, es decir, al sistema de archivos y carpetas, que es lo que se conoce como **File System**. Eso es lo que haremos en este ejercicio, utilizando un módulo llamado `fs`, que contiene funcionalidades como crear, leer y editar archivos.

Pauta:

1. Crear una carpeta llamada `ejercicio_filesystem`.
2. Inicializar un proyecto con el comando `npm init`.
Pueden ignorar todas las preguntas haciendo “Enter” en cada una.
3. Verificar que se haya creado el archivo `package.json`.
4. Crear un archivo: `index.js`.
Hacer `console.log(__dirname)`; en el archivo para ver qué retorna.
5. Importar (requerir) el módulo `fs`.
6. Crear un servidor escuchando en el puerto 8000.
7. Utilizar el método `appendFile` del módulo `fs` para crear un archivo llamado `access_log.txt`. El archivo se debe crear en el primer llamado al servidor y para cada nuevo llamado se debe crear un línea de texto que contenga la fecha actual, con este formato:

```
Se llamó al servidor el 17 de marzo de 2021 a las 21:56:57 (martes).
Se llamó al servidor el 17 de marzo de 2021 a las 22:29:58 (martes).
Se llamó al servidor el 17 de marzo de 2021 a las 22:29:58 (martes).
Se llamó al servidor el 17 de marzo de 2021 a las 22:30:04 (martes).
Se llamó al servidor el 17 de marzo de 2021 a las 22:30:04 (martes).
Se llamó al servidor el 17 de marzo de 2021 a las 22:30:11 (martes).
Se llamó al servidor el 17 de marzo de 2021 a las 22:30:11 (martes).
```

Podría ser útil instalar la librería [date-fns](#) [Moment](#) para manipular las fechas con mayor facilidad.

👉 Al finalizar el ejercicio, pedirle a un **compañero** (alguien con el que no hayan trabajado antes) que les **corrija/revise** el código y viceversa. Además de que el código funcione, verificar que esté prolijo y entendible.

Ejercicio 7

Clase previa: “Node/npm”.

Pauta:

Crear una pequeña aplicación en Node.js que recibe dos argumentos:

1. La ruta a un directorio. Ej: “/Users/Maria/Documentos”.
2. Una extensión de archivo. Ej: “txt”.

El archivo se debe llamar `listadoFiltrado.js`.

La forma de llamarlo a través de la **consola** será así:

```
node listadoFiltrado.js "/Users/Maria/Documents" "txt"
```

Esto debe retornar como resultado (en la Terminal) algo similar al siguiente mensaje:

```
Archivos encontrados:
tareas.txt
cv.txt
maria_lopez.txt
```

En caso de no encontrarse nada, mostrar un mensaje de error.

Deberán investigar sobre:

- [process.argv](https://nodejs.org/api/process.html).
- También puede ser útil el módulo [path](https://nodejs.org/api/path.html).

Ejercicio 8

Clase previa: “Express”.

Pauta:

Crear un sitio web con **Express** que contenga las siguientes rutas:

- [GET] <http://localhost:3000>.
- [GET] <http://localhost:3000/productos>.
- [GET] <http://localhost:3000/sobre-nosotros>.
- [GET] <http://localhost:3000/contacto>.

Crear un cabezal (ej: usando el componente navbar de Bootstrap) que aparezca en todas las páginas y que contenga links entre ellas. Es probable que el código del cabezal lo tengan que repetir en cada página (por ahora no hay problema con eso; dejarlo repetido).

Levantar el servidor en el puerto 3000 y verificar que el sitio sea correctamente navegable.

Para enviar un archivo como parte de una respuesta HTTP pueden usar el método [sendFile](#).

Ejercicio 9

Clase previa: “Express”.

Pauta:

Crear un sitio web con **Express** que contenga las siguientes rutas:

- [GET] <http://localhost:3000>.
- [GET] <http://localhost:3000/multiplicar>.

En la Home debe haber un formulario HTML que contenga dos campos:

- Número 1.
- Número 2.

El formulario debe tener un botón llamado “**Multiplicar**” y al hacer click sobre el mismo se debe llamar a la ruta `/multiplicar`, a la cual se le deben pasar los números por el *query string*. Ejemplo:

<http://localhost:3000/multiplicar?num1=5&num2=6>

Al acceder a dicha URL debería aparecer el texto: “El resultado es 30”. Esta nueva página no debe ser un HTML.

Ingresa dos números para multiplicar

Número 1

Número 2

Multiplicar

Ejercicio 10

Clase previa: “Express”.

Pauta:

Modificar el ejercicio anterior para que en lugar de ver el resultado en una nueva página, el mismo se vea debajo del formulario, luego de hacer click en el botón. No se deberá recargar la página al presionar el botón.


Se deberá seguir utilizando la URL <http://localhost:3000/multiplicar> para realizar el cálculo (la multiplicación), por lo que será necesario hacer una llamada **AJAX** desde la página del formulario y recibir el resultado a mostrar.

Ingresa dos números para multiplicar

Número 1

Número 2

El resultado es: 24

 **Nota:** verificar que la multiplicación se realice del lado del servidor (*back-end*) y no en el navegador.

Ejercicio 11

Clase previa: “EJS”.

Pauta:

1. Crear una carpeta llamada `ejercicio_ejs`.
2. Inicializar un proyecto con el comando `npm init`.
Pueden ignorar todas las preguntas haciendo “Enter” en cada una.
3. Verificar que se haya creado el archivo `package.json`.
4. Crear un archivo: `index.js`.
5. Instalar los módulos `express` y `ejs`. Tener `nodemon` instalado de forma global.
6. Crear las siguientes rutas:
 - [GET] <http://localhost:3000>.
 - [GET] <http://localhost:3000/productos>.
 - [GET] <http://localhost:3000/sobre-nosotros>.
 - [GET] <http://localhost:3000/contacto>.
7. Crear un cabezal (ej: usando el componente navbar de Bootstrap) que aparezca en todas las páginas y que contenga links entre ellas. El código del cabezal debe escribirse sólo una vez, para **evitar código repetido** y lograr un sitio más **mantenible**.
8. Levantar el servidor en el puerto 3000 y verificar que el sitio sea correctamente navegable.
9. En la Home (debajo del cabezal) debe haber un texto que diga:
 - “Hoy es un día de semana”, si estamos entre Lu y Vi.
 - “Hoy es fin de semana”, si estamos en Sábado o Domingo.
10. A la vista de productos, pasarle un array de strings con nombres de productos (ej: "Notebook", "Impresora", "Monitor", etc), el cual se deberá mostrar en una lista ``.

👉 Al finalizar el ejercicio, pedirle a un **compañero** (alguien con el que no hayan trabajado antes) que les **corrija/revise** el código y viceversa. Además de que el código funcione, verificar que esté prolijo y entendible.

Ejercicio 12

Clase previa: “Middlewares”.

Pauta:

1. Crear una carpeta llamada `ejercicio_listaFrutas`.
2. Inicializar un proyecto con el comando `npm init`.
Pueden ignorar todas las preguntas haciendo “Enter” en cada una.
3. Verificar que se haya creado el archivo `package.json`.
4. Crear un archivo: `index.js`.
5. Instalar los módulos `express` y `ejs`. Tener `nodemon` instalado de forma global.
6. Crear las siguientes rutas:
 - [GET] <http://localhost:3000/frutas>.
 - [POST] <http://localhost:3000/frutas>.
7. En la página de frutas debe mostrarse:
 - Un título `<h1>`.
 - Un listado `` (inicialmente con 3 frutas: "Manzana", "Pera" y "Frutilla", es decir, con 3 elementos ``).
 - Un **formulario** con un campo de texto y un botón llamado “Agregar”.
8. El `action` del formulario debe ser la URL anterior y el **método POST**.
9. Al hacer click en el botón se debe mostrar la misma página (recargada), pero con la nueva fruta.

👉 Probablemente será necesario que a nivel de Back-End, las frutas estén guardadas en un **array**. Por lo tanto, notar que cuando se haga un reinicio de la aplicación de Node.js se perderán todas las frutas que habían en la lista. Está bien que así sea (para este ejercicio).

También podrán notar que, mientras no se reinicie la aplicación, si entran a la página desde otra ventana de navegador (u otro dispositivo), verán la misma lista de frutas.

Ejercicio 13

Clase previa: “Middlewares”.

Pauta:

 Ver la charla que **Ryan Dahl** dio en 2009, dando a conocer Node.js. Link:

<https://www.youtube.com/watch?v=ztspvPYyblY>

Son como 50 minutos pero valen la pena. Ayuda a entender el momento histórico y el porqué del surgimiento de Node.js.

Ejercicio 14

Clase previa: “Middlewares”.

Pauta:


1. Crear una carpeta llamada `ejercicio_express_json`.
2. Inicializar un proyecto con el comando `npm init`.
Pueden ignorar todas las preguntas haciendo “Enter” en cada una.
3. Verificar que se haya creado el archivo `package.json`.
4. Crear un archivo: `index.js`.
5. Instalar el módulo `express`.
6. Crear un archivo llamado `db.js`, que simulará ser una base de datos. En dicho archivo, colocar el contenido de este *gist*: [link](#).
7. Crear las siguientes rutas (*endpoints*):
 - [GET] `/teams` para obtener todos los *teams* en la “base de datos”.
 - [GET] `/teams/:id` para obtener aquel *team* cuyo *id* matchee con el presente en la ruta del *request*.
 - [POST] `/teams` para insertar en la “base de datos” el *team* que pasaremos en el *body* del *request* (en formato JSON). Ejemplo del *body*:

```
{
  "id": "uy",
  "name": "Uruguay",
  "flag": "🇺🇾"
}
```
 - [DELETE] `/teams/:id` para remover de la “base de datos” aquel *team* cuyo *id* matchee con el presente en la ruta del *request*.
 - [PATCH] `/teams/:id` para alterar, con los datos que incluyamos en el *body* del *request* (en formato JSON), aquel *team* cuyo *id* matchee con el presente en la ruta del *request*.
8. Para **probar** las llamadas, no se podrá usar un navegador, ya que hay métodos que no son soportados por los mismos. Deberán instalar un programa como [Insomnia](#) o [Postman](#), y hacer los llamados desde allí.

👉 Al finalizar el ejercicio, pedirle a un **compañero** (alguien con el que no hayan trabajado antes) que les **corrija/revise** el código y viceversa. Además de que el código funcione, verificar que esté prolijo y entendible.

Ejercicio 15

Clase previa: “Middlewares”.

La idea de este ejercicio es crear un formulario que permita a nuestros usuarios suscribirse a nuestro **newsletter**. Para eso usaremos una herramienta llamada **Mailchimp** .

Mailchimp permite almacenar una lista de contactos a los cuales se le enviarán los newsletter. Por lo tanto, Mailchimp no sólo funciona para enviar correos sino que también funciona, en cierto modo, como una **base de datos** de usuarios.

Pauta:

1. Crear una carpeta llamada `ejercicio_mailchimp`.
2. Inicializar un proyecto con el comando `npm init`.
Pueden ignorar todas las preguntas haciendo “Enter” en cada una.
3. Verificar que se haya creado el archivo `package.json`.
4. Crear un archivo: `index.js`.
5. Instalar los módulos `express` y `ejs`.
6. Crear las siguientes rutas (*endpoints*):
 - [GET] <http://localhost:3000/newsletter-registro>.
 - [POST] <http://localhost:3000/newsletter-registro>.
 - [GET] <http://localhost:3000/newsletter-gracias>.
 - [GET] <http://localhost:3000/newsletter-error>.
7. La página de registro debe contener un pequeño formulario HTML con:
 - Campo de texto para un nombre.
 - Campo de texto para un apellido.
 - Campo de texto para un email.
 - Botón con el texto “Registrarme”.
 - El formulario debe aparecer centrado en la página.
 - El action del formulario debe ser `/newsletter-registro` y el método POST.
8. Crearse una cuenta en [Mailchimp](#) y darle una vichada a la [documentación para desarrolladores](#). Van a tener que conseguir una **API Key** (que es una

especie de contraseña para poder acceder a la **Marketing API** de Mailchimp).

9. También deberán crear un Audience (Audiencia), la cual tendrá asociada una **Audience Id**. Este es un dato que necesitarán para el siguiente punto.

Nota 1: Esta Audiencia se podrá crear “a mano” desde el dashboard de Mailchimp (no es necesario utilizar la API).

Nota 2: En la documentación de Mailchimp, a las **audiencias** también se les llama **listas**.

10. Desde el Back-End (desde Node.js) se debe **llamar a la API** de Mailchimp y pasarle los datos del usuario que se quiere anotar en el newsletter. Esto lo podrán hacer de diversas maneras. Node.js ya trae un módulo llamado `https` que les puede servir (aunque es de “bajo nivel”). De lo contrario pueden instalar algún **módulo** de un tercero como [Axios](#). **Actualización**: En agosto de 2020, Mailchimp lanzó una nueva versión de su API e incluyó una [pequeña librería para usar con Node.js](#). Esto simplifica mucho el llamado a la API.
11. Agregar validación al formulario de registro. Es decir, validar que los datos ingresados sean válidos, particularmente el email y que el nombre y apellido tengan al menos dos letras cada uno. La validación se debe realizar del lado del Back-End (antes de enviar los datos a Mailchimp). Para realizar la validación pueden usar algún **módulo** de un tercero.
12. Si además quieren validar los datos del lado del Front-End, ¡genial!.

Ejercicio 16

Clase previa: “Middlewares”.

Pauta:

1. Crear una carpeta llamada `ejercicio_blog`.
2. Inicializar un proyecto con el comando `npm init`.
Pueden ignorar todas las preguntas haciendo “Enter” en cada una.
3. Verificar que se haya creado el archivo `package.json`.
4. Crear un archivo: `index.js`.
5. Instalar los módulos `express` y `ejs`.
6. Crear las siguientes rutas (*endpoints*):
 - [GET] <http://localhost:3000>.
 - [GET] <http://localhost:3000/articulos/:id>.
7. La Home debe mostrar todos los artículos disponibles en [este gist](#). Para eso se deberá hacer una llamada HTTP a la URL del *gist*, desde el Back-End (lo cual se podría hacer con una librería externa como [Axios](#)). De cada artículo se deberá mostrar:
 - Título.
 - Pequeño texto (ej: hasta 50 caracteres). Si un texto queda “cortado a la mitad”, se pueden poner puntos suspensivos (...).
 - Botón / Link a la página del artículo.
8. Además, cada artículo deberá tener su propia página. Por ejemplo, para acceder al artículo con ID = 7 se deberá acceder a la siguiente URL:
<http://localhost:3000/articulos/7>. En esta página se deberá ver la información completa de cada artículo:
 - Título.
 - Texto completo.
 - Imagen.
 - Autor.
 - Botón / Link para regresar a la Home.

Ejercicio 17

Clase previa: “SQL”.

Pauta:

Completar el siguiente entrenamiento sobre SQL: <https://www.sqlteaching.com>.

Ejercicio 18

Clase previa: “SQL”.

Pauta:

- Instalar MySQL utilizando el material compartido en clase.
- Instalar alguna GUI de su preferencia (ej: TablePlus o MySQL Workbench), a no ser que se sientan más cómodos trabajando con la CLI.
- Crear una base de datos llamada `db_test`. Seleccionar como cotejamiento (encoding): `utf8mb4_unicode_ci`. Leer más al respecto [aquí](#).
- Crear una tabla llamada “usuarios”.
- Crear las siguientes columnas:
 - `id` (BIGINT, auto-incremental, clave primaria).
 - `nombre` (VARCHAR, longitud 100).
 - `apellido` (VARCHAR, longitud 100).
 - `edad` (TINYINT).
 - `password` (VARCHAR, longitud 150).
- Todos los datos deben ser obligatorios.
- Insertar 2 usuarios en la base de datos. Verificar que los datos hayan sido ingresados correctamente. ¿Qué se puede decir sobre guardar el campo `password` de esta forma?

Ejercicio 19

Clase previa: “Sprint 1”.

Notar que este ejercicio no está ligado con los ejercicios anteriores, es simplemente para seguir practicando sobre **Git**, tema visto en el Sprint 1.

Pauta:

Completar los ejercicios de esta página: <https://learngitbranching.js.org>.

Ejercicio 20

Clase previa: “MySQL y Node”.

Pauta:

1. Crear una carpeta llamada `ejercicio_mysql`.
2. Inicializar un proyecto con el comando `npm init`.
Pueden ignorar todas las preguntas haciendo “Enter” en cada una.
3. Verificar que se haya creado el archivo `package.json`.
4. Crear un archivo: `index.js`.
5. Instalar los módulos `express`, `mysql2` y `ejs`.
6. Crear una base de datos MySQL llamada `ha_ejercicio_20`.
7. Crear una tabla llamada `users`.
8. Crear las siguientes columnas:
 - `id` (BIGINT, auto-incremental, PK).
 - `firstname` (VARCHAR, 100, Not Null).
 - `lastname` (VARCHAR, 100, Not Null).
 - `age` (INT).
9. Insertar en la tabla el contenido de [este gist](#).
10. Replicar una página como la que se ve a continuación.
11. Crear las siguientes rutas (*endpoints*):
 - [GET] [/usuarios](#) → Retorna un HTML (muestra la tabla de usuarios).
 - [GET] [/usuarios/crear](#) → Retorna un HTML con un formulario de creación.
 - [GET] [/usuarios/editar/:id](#) → Retorna un HTML con un formulario de edición..
 - [POST] [/usuarios](#) → Crea un nuevo usuario en la BD.
 - [POST] [/usuarios/editar/:id](#) (Por ahora será POST, aunque idealmente debería ser un PATCH).
 - [GET] [/usuarios/eliminar/:id](#) (Por ahora será GET, aunque idealmente debería ser un DELETE).

12. Hacer funcionar los formularios de Crear y Modificar usuario, así como el botón Eliminar. Todos los cambios deben verse afectados en la base de datos.

13. En este ejercicio no será necesario usar AJAX.

Referencias visuales:

Listado de Usuarios

Nuevo

Id	Nombre	Apellido	Edad	Acciones
4	Juan	Fagúndez	44	<div>EditarEliminar</div>
6	Sofía	Jiménez	31	<div>EditarEliminar</div>
7	Pablo	Gutiérrez	17	<div>EditarEliminar</div>
8	Victoria	Gómez	31	<div>EditarEliminar</div>
9	José	Gómez	20	<div>EditarEliminar</div>
10	María Laura	Jiménez	30	<div>EditarEliminar</div>
11	Juan Pablo	Rodríguez	46	<div>EditarEliminar</div>
12	Inés	Vázquez	29	<div>EditarEliminar</div>

Crear nuevo usuario

Nombre

Ingresar nombre...

Apellido

Ingresar apellido...

Edad

Ingresar edad...

Guardar

[← Volver](#)

Modificar usuario

Nombre

Juan

Apellido

Fagúndez

Edad

44

Modificar

[← Volver](#)

En este ejercicio se está haciendo lo que comúnmente se conoce como **CRUD** de una entidad (en este caso la entidad Usuario):

- C – Create.
- R – Read.
- U – Update.
- D – Delete.

Ejercicio 21

Clase previa: “MySQL y Node / Sequelize”.

La idea de este ejercicio es hacer un **Blog**, es decir, un sitio web donde se muestren artículos (noticias) como, por ejemplo, [TechCrunch](#).

Es un ejercicio largo y deberán **trabajar en equipo** (armados previamente por el docente y comunicados por Slack/Teams). Deberán **dividirse las tareas** tratando de que cada integrante realice aproximadamente la misma cantidad de trabajo e intentando que las mismas sean variadas (es decir, no vale que uno haga todo el Front-End y otro haga todo el Back-End). Tal vez les puede ser útil apoyarse en una herramienta como [Trello](#) o usar la herramienta de gestión de proyectos integrada en [GitHub](#).

👉 Cuiden mucho la **comunicación** entre ustedes. Además de tratarse con respeto, mantengan informados a sus compañeros. Júntense físicamente y/o hagan videollamadas donde se cuenten sobre lo que están haciendo, en qué están trancados y qué planean hacer. La buena comunicación en el equipo es clave para el **éxito del proyecto**. Por favor, dediquen tiempo para hacer una buena **planificación**.

Deberán tener un repositorio privado en **GitHub** donde compartirán su código.

Es posible que para construir ciertas partes del Blog deberán **investigar temas por su cuenta** o esperar a alguna clase teórica del Bootcamp.

Hay partes de la pauta que no están especificadas al 100%, por lo que el equipo deberá tomar ciertos supuestos y/o hacerle preguntas al docente, que oficiará de Product Owner en el proyecto.

El equipo también deberá ser responsable de **definir cómo organizar su código**, elegir nombres para las URLs, elegir nombres para tablas/columnas de la base de datos, etc. Cuidar mucho la **prolijidad** del código; hacerlo **entendible** y **mantenible**.

Pauta:

1. Crear una carpeta llamada `ejercicio_blog`.
2. Realizar setup inicial e instalar dependencias necesarias.

3. El Blog debe tener las siguientes funcionalidades:
 - 3.1. Tener una **Home** donde se pueda ver el **listado total** de artículos. Deben aparecer ordenados por fecha de creación. El más nuevo, arriba del todo.
 - 3.2. Cada artículo debe tener su **propia URL** para poder ver detalles sobre el mismo.
 - 3.3. De cada **artículo** se debe conocer:
 - 3.3.1. Título.
 - 3.3.2. Contenido.
 - 3.3.3. Imagen.
 - 3.3.4. Fecha de creación.
 - 3.3.5. Autor (del cual se deba conocer el nombre, apellido y email).
 - 3.3.6. Listado de comentarios (ya que se podrán dejar comentarios en la página del artículo).
 - 3.4. Debe haber una sección del Blog que funcione como **Administrador** (Admin), desde donde se podrán crear, modificar y eliminar artículos. Por ahora no hay que preocuparse de realizar ningún tipo de autenticación, es decir, cualquier navegante que acceda a las URLs correspondientes podrá gestionar los artículos del Blog. Se sugiere que todas las URL relativas al Admin estén bajo la URL:
<http://localhost:3000/admin>.
 - 3.5. No olvidar hacer validaciones de los datos ingresados por el usuario, así como mostrar **mensajes de error** correspondientes.
 - 3.6. Cada vez que alguien cree un artículo en el Blog se debe enviar un **email** a cada integrante del equipo notificando lo ocurrido.
 - 3.7. Los artículos del Blog también deben quedar disponibles en formato JSON a través de una ruta del tipo: <http://localhost:3000/api/articulos>.
4. No hay requisitos particulares en cuanto a cómo deberá ser la estética del Blog. Esto deberá ser definido por el equipo.
5. Se deberán crear artículos en la base de datos de forma que haya **contenido inicial** en el Blog (para que no esté todo vacío al principio).
6. [Extra] Crear una cuenta de [Google Analytics](#) y agregarla al sitio para poder *trackear* las vistas en todas las páginas.

Ejercicio 22

Clase previa: “SQL”.

Pauta: Entrenamiento extra sobre SQL:

- <https://sqlbolt.com>.
- <https://hunter-ducharme.gitbook.io/sql-basics>.
- <https://flashcards.github.io/sql/introduction.html>.
- https://sqlzoo.net/wiki/SQL_Tutorial.

Ejercicio 23

Clase previa: “Clean Code & MVC”.

Pauta:

- Darle una leída a [Idiomatic.js](http://idiomatic.js).
- Investigar sobre SOLID:
 - **S**ingle Responsibility Principle (SRP).
 - **O**pen/Closed Principle (OCP).
 - **L**iskov Substitution Principle (LSP).
 - **I**nterface Segregation Principle (ISP).
 - **D**ependency Inversion Principle (DIP).
- Investigar sobre patrones de diseño. Algunos particularmente interesantes son:
 - MVC.
 - Observer.
 - Singleton.
 - Facade.
 - Factory Method.