



0.- Índice

1.- Introducción	Página 3
2.- Objetivos	Página 4
3.- Justificación	Página 7
4.- Plan de trabajo	Página 8
4.1.- Entorno PHP	Página 8
4.2.- Aplicación Chat	Página 13
4.3.- Heroku y Github	Página 26
5.- Calendario de acciones	Página 31
6.- Evaluación de las acciones del proyecto	Página 32
7.- Conclusiones finales y líneas de futuro	Página 33
8.- Bibliografía	Página 34

1.- Introducción

A lo largo de este documento explicativo, se desarrollarán los pasos, dificultades y necesidades surgidas durante el proceso de creación de un Chat Web basado en Node (javascript).

El proyecto en su totalidad se ha enfocado al ámbito de la programación, desarrollar el producto más simple pero eficiente posible en un lenguaje de programación desconocido hasta la fecha, pero sin dejar de lado los conocimientos adquiridos durante el ciclo. Así pues, ámbitos como los estilos (css) o la seguridad se han reducido al mínimo, como digo, para centrarme mayoritariamente en aprender un nuevo lenguaje de programación e implementarlo al que sí hemos sido instruidos (PHP).

La idea de proyecto surge desde la propia empresa en la que estoy de prácticas, múltiples comentarios por parte de mis compañeros de trabajo haciendo referencia a la necesidad de una plataforma de comunicación interna, teniendo que recurrir ellos a aplicaciones existentes como Whatsapp o Telegram sobre las que no se tiene control real.

Hoy, habiendo terminado el producto final, se me incita por parte de la empresa a continuar expandiendo el proyecto para transformarlo de un chat realmente simple (pero que sirve de base) a una aplicación de comunicación aplicable a la sucursal.

2.- Objetivos

Mis objetivos fundamentales a la hora de plantear el proyecto fueron:

- Aprender programación básica orientada a la comunicación en JS.
- Crear una aplicación Chat utilizando los nuevos conocimientos adquiridos.
- Integrar la aplicación en un entorno WEB mayoritariamente PHP.
- Hospedar la aplicación Chat en Heroku (del que hablaré más adelante).
- Determinar pasos futuros.

2.1.- Aprender programación básica orientada a la comunicación en JavaScript:

→ Como para aprender cualquier lenguaje nuevo, en programación pasa lo mismo. Hay que estudiar, buscar información, y estudiar más, por lo que gran parte del proyecto ha consistido en obtener los conocimientos necesarios para desarrollar una aplicación Chat en un lenguaje del que no se tiene base sólida **e implementarlo a los conocimientos adquiridos durante ciclo.**

2.2.- Crear una aplicación utilizando los nuevos conocimientos adquiridos:

→ La aplicación Chat constará de un único archivo HTML mediante el cual el usuario podrá introducir un nombre de usuario alternativo (independientemente del que haya utilizado para registrarse en el sitio), y conversar con aquellos usuarios que estén conectados simultáneamente.

Éste archivo HTML se vinculará en tiempo real con otro en JavaScript que **no necesita estar hospedado en la misma máquina** y que será realmente el que gestione los datos, acciones y eventos. Será nuestro servidor de chat, básicamente.

2.3.- Integrar la aplicación en un entorno WEB mayoritariamente PHP:

→ Aunque durante el planteamiento del proyecto éste sería el tercer objetivo, terminará por convertirse en el segundo a la hora de llevarlo a la práctica para facilitar la posterior integración del chat JavaScript en un entorno PHP.

→ Se creará un servidor WEB haciendo uso de los conocimientos adquiridos durante el curso. Se montará un servidor XAMPP en Windows 10.

→ Se gestionarán bases de datos MySQL, tanto desde la interfaz phpMyAdmin como desde el código de la aplicación en sí. Contaremos con una única base de datos (proyecto) dividida en dos tablas (webchat_users y webchat_lines)

→ El sitio PHP constará únicamente de cuatro archivos:

1.- index.php

Donde el usuario introduce sus datos para entrar al sitio, o donde decide registrar una cuenta nueva.

2.- regUser.php

Donde el usuario que ha decidido registrar una cuenta nueva introducirá sus datos.

3.- db.php

Clase que construye y prepara la conexión a la base de datos, para incluirla en aquellos archivos que la necesiten y evitar abrir y cerrar conexiones individuales.

4.- checkUser.php

Aquí es donde todas las comprobaciones se llevan a cabo, este archivo gestionará toda la información, tanto de intentos de login como de registro, y la cotejará o añadirá a la base de datos.

2.4.- Hospedar la aplicación Chat en Heroku:

→ Heroku es una plataforma de computación en la Nube, capaz de procesar diversos lenguajes de programación. En mi caso, es capaz de ejecutar aplicaciones basadas en Node de manera continua, y lo más importante, gratuitamente (hasta cierto punto).

Entonces, la idea es hospedar únicamente la aplicación de Chat en Heroku, para integrarla posteriormente al resto del proyecto hospedado de forma local.

Heroku además, puede vincularse con un repositorio de Github, o sea que únicamente tengo que vincular mi propio repositorio con una cuenta de Heroku.

2.5.- Determinar pasos futuros

→ Una vez cumplidos todos los puntos previos, se pretende estudiar de qué manera o maneras podría mejorarse la aplicación, ya no como proyecto si no como producto, en mi caso, de empresa.

3.- Justificación

Durante la duración del ciclo, e incluso antes, un amigo y yo ya tonteábamos con la idea de crear nuestro propio entorno de comunicación para evadir los comerciales o, en general, aquellos que no podemos controlar en su totalidad. Nace entonces la curiosidad que empuja a cualquier programador a aprender: ¿Cómo funciona? ¿Será realmente tan seguro como se plantea? ¿Será algo que esté realmente tan fuera del alcance de alguien inexperto?

Aunque seguramente ninguno de los dos podamos afirmar con firmeza que la idea surgiera de intenciones nobles, sí nos ha terminado por impulsar a ambos hacia el mismo destino, el interés por la programación y la seguridad en la red.

Más adelante, ya de prácticas en la empresa, la idea se vio reforzada ante la necesidad real de una plataforma propia mediante la que poder conversar libremente sobre lo que fuere. Es aquí donde surge realmente la idea de proyecto, crear una aplicación de chat web que, aunque seguramente muy simple, sirva de base para expansiones futuras y, sobre todo, enfocada a la programación. Nunca he sido buen artista.

4.- Plan de trabajo

Por motivos de simplicidad, separaré las secciones del proyecto a explicar en dos: Login y Registro (que hace referencia al contenido PHP, el entorno web) y Chat (que es la aplicación de mensajería integrada).

4.1- Login y Registro (PHP)

Como ya mencioné anteriormente, se montó un servidor XAMPP bajo Windows 10 para ejecutar el entorno WEB, y como editor de código se escogió Visual Estudio Code por su versatilidad y funcionalidad. Aunque más adelante se editarían documentos directamente desde Github, para ser capaz de realizar cambios tanto desde el trabajo como desde casa. Además, hospedar las aplicaciones en Github nos allana el terreno para lanzarlas luego a Heroku.

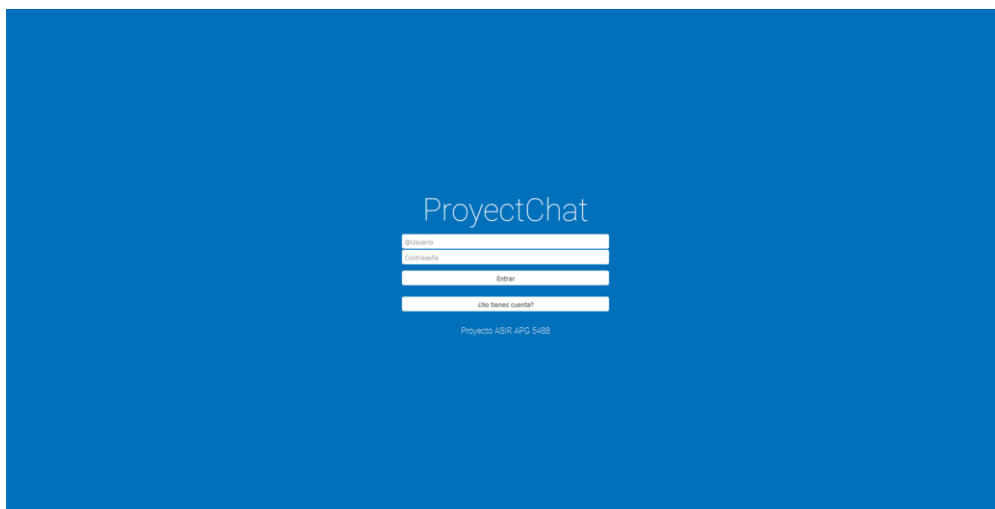


La parte PHP del proyecto constará únicamente de cuatro archivos PHP/HTML que serán explicados a continuación:

[index.php](#)

Archivo principal al que se accede, muestra un simple formulario de acceso al sitio, obligando al usuario a introducir sus datos y respondiendo a los mismos de diferentes maneras.

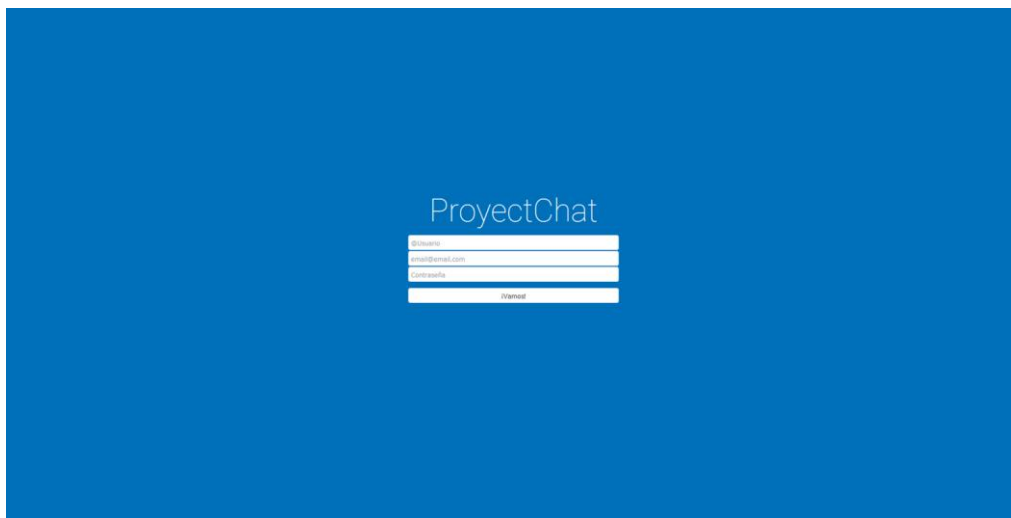
Incluye dos botones que redireccionan a otros dos de los archivos PHP: checkUser.php y regU-ser.php



[regUser.php](#)

Se trata de un formulario alternativo para aquellos usuarios que se quieran crear una cuenta, habiendo pulsado el botón “¿No tienes cuenta?” en la página anterior. Nuevamente, se mostrarán campos de información a cubrir por el usuario y un único botón para enviarlos.

Tanto en este archivo como en el anterior, la información introducida es enviada y procesada por checkUser.php.

A screenshot of a web registration form titled "ProjectChat" centered on a solid blue background. The form consists of four white input fields stacked vertically, each with a light blue border. The labels for the fields are "Nombre", "Email", "Contraseña", and "Repetir", positioned to the left of each input field. Below the "Repetir" field is a small, light blue button with the text "Enviar" in white.

[checkUser.php](#)

A diferencia de los anteriores, no presenta contenido gráfico o html, se trata de únicamente programación lógica en PHP. Determina de donde viene la información (si de un intento de acceso o de registro) y la coteja con la base de datos, comprobando si existe, si es correcta, si está disponible, etc. Una vez procesada la información asigna variables de sesión que son devueltas a los archivos originales para hacer las páginas reactivas.

Por ejemplo, si introduzco un usuario que no existe al intentar acceder, el programa me avisará resaltando el campo en rojo, y cambiando el texto de los botones.

The image displays three screenshots of the 'ProyectoChat' login interface, which has a blue background and white text. Each screenshot shows the title 'ProyectoChat' at the top and 'Proyecto ASIR APG 5488' at the bottom.

- Top Left Screenshot:** Shows the login form with the '@Usuario' field highlighted in red. Below the password field, the text 'Usuario Inexistente' is displayed. The 'Entrar' button is disabled.
- Top Right Screenshot:** Shows the login form with the 'Contraseña' field highlighted in red. Below the password field, the text 'Contraseña Incorrecta' is displayed. The 'Entrar' button is disabled.
- Bottom Center Screenshot:** Shows the login form with both fields filled. The 'Entrar' button is active and highlighted in green, with the text '¡Bienvenido!' displayed below it.

db.php

De nuevo un archivo únicamente lógico, en el que se construyen las conexiones a la base de datos y así evitar crearlas y destruirlas en cada archivo PHP de manera individual. Este archivo se incluye en checkUser.php, por lo que la información tratada en uno, se trata en ambos.

```

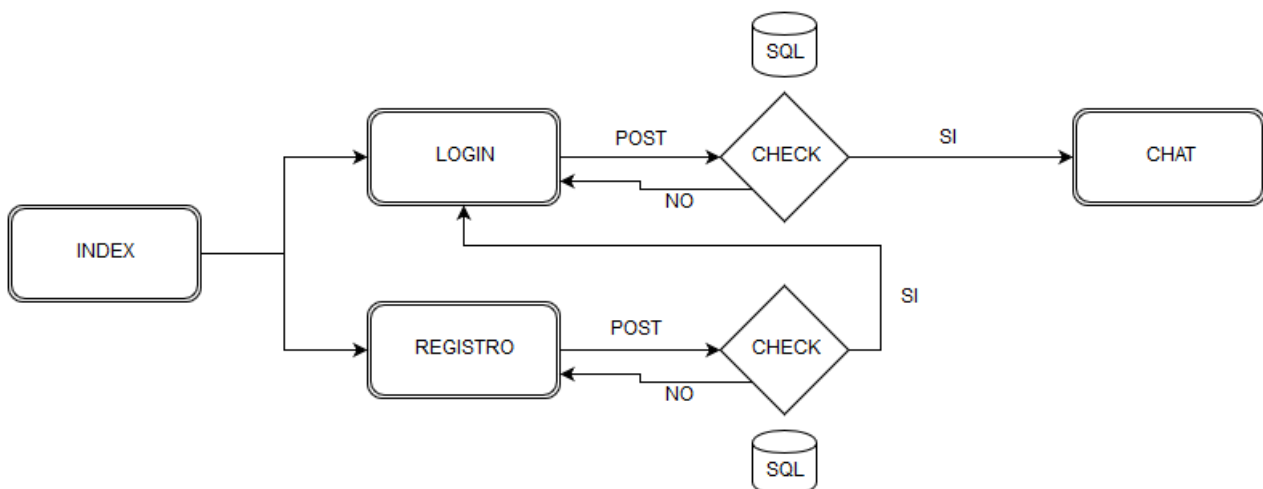
?php
// Simple conexión PDO (PHP Data Object) a la base de datos y el archivo php. Para posterior uso en checkUser.php

$dbhost    = 'localhost'; // Donde se hospeda la base de datos
$dbuser    = 'root';      // Nombre de usuario para la base de datos
$dbpass    = '';          // Contraseña para la base de datos
$dbname    = 'proyecto';  // Base de datos a seleccionar

$connection = new mysqli($dbhost, $dbuser, $dbpass, $dbname); // Conexión a la base de datos atendiendo a los valores definidos ^
?

```

Esto concluye la parte PHP del proyecto hospedada, recuerdo en XAMPP bajo Windows 10.



4.2- Chat

A continuación, explicaré en profundidad el proceso de creación de la aplicación de chat javascript, así como todos los requisitos previos.

4.2.1.- Introducción

He comentado ya varias veces que la aplicación fue o sería programada en javascript pero, como bien se sabe, es un lenguaje de programación que normalmente va implementado del lado cliente, como parte del navegador, y que se usa para añadir funcionalidades gráficas a los sitios web, así como dinámico.

Sin embargo, para la creación de una aplicación de chat no nos basta con interactuar con los clientes. Necesitamos un servidor que procese la información y tome decisiones. Entonces, ¿Es Javascript realmente el lenguaje de programación que necesitamos? Bueno, no por sí solo al menos.

Aquí es donde entra en acción **NODE** (node.js), que es un entorno de ejecución multiplataforma pensado para servidores (aunque no limitado a ello) orientado a eventos y basado en ECMAScript que a su vez está basado en Javascript.



4.2.2.- Preparación y dependencias

Una vez hayamos descargado e instalado node.js apropiadamente, contaremos con una herramienta nueva en nuestro equipo: NPM.

NPM (Node Package Manager) es el gestor de paquetes por defecto de node.js, que nos permitirá instalar, de manera sencilla, todas las librerías y complementos que necesitemos.

Crearemos pues un nuevo directorio para nuestro proyecto, abriremos un terminal de comandos y ejecutaremos “*npm init*”.

Esto nos guiará hacia la creación de un archivo json vinculado a nuestro proyecto que contendrá información como el nombre de la aplicación, su versión y, lo más importante, sus dependencias. Esto último es importante porque más adelante será necesario para hospedar la aplicación en Heroku.

```
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See 'npm help json' for definitive documentation on these fields
and exactly what they do.

Use 'npm install <pkg>' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (proyecto)
version: (1.0.0)
description: Proyecto Asir
entry point: (index.js)
test command:
git repository:
keywords:
```

```
{ } package.json x
1  {
2    "name": "proyecto",
3    "version": "1.0.0",
4    "description": "Proyecto Asir",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "Alberto Paz Garcia",
10   "license": "ISC"
11  }
12
```

El siguiente paso que debemos llevar a cabo es importar los módulos de node.js al directorio de nuestro proyecto. Para ello ejecutaremos el comando `"npm install --save node.js"`.

Si el resultado ha sido satisfactorio, veremos aparecer una nueva carpeta en nuestro directorio con el nombre de `"node_modules"`.

A estas alturas, estamos listos para comenzar a instalar todas las librerías que vamos a utilizar para la creación de la aplicación. Nótese que si se hace correctamente, el archivo package.json mostrado anteriormente se actualizará automáticamente para listar las dependencias y sus respectivas versiones.

Para llevar a cabo mi proyecto, instalé y utilicé las siguientes librerías:

- express → Infraestructura WEB
- socket.io → Comunicación bidireccional en tiempo real.
- mysql → Gestión y acceso a bases de datos desde Javascript.

```
$ npm install --save express
```

```
$ npm install --save socket.io
```

```
$ npm install --save mysqljs/mysql
```

```
1  {
2    "name": "proyecto",
3    "version": "1.0.0",
4    "description": "Proyecto Asir",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "Alberto Paz Garcia",
10   "license": "ISC",
11   "dependencies": {
12     "express": "^4.16.2",
13     "mysql": "github:mysqljs/mysql",
14     "node.js": "0.0.0",
15     "socket.io": "^2.0.4"
16   }
17 }
18
```

4.2.3.- Servidor

A continuación, cubriremos la creación del script que se ejecutará a modo de servidor, así como los eventos a los que atenderá.

Una vez incluidas todas las dependencias que se van a utilizar, lo primero que se lleva a cabo es definir en qué puerto escuchará el servidor, así como la creación de una conexión a la base de datos con la que se va a comunicar.

```
server.listen(process.env.PORT || 3000); // Puerto en el que va a escuchar el servidor
console.log('Server Runing...');

var con = mysql.createConnection({ // Constructor para las conexiones a la base de datos.
  host:      "localhost",
  user:      "root",
  password:  "",
  database:  "proyecto"
});

if(con.connect(function(err){ // Intento de conexión a la base de datos ^
  if(err) throw err;
  console.log("MySQL Connected Successfully...");
})));
```

Si todo va bien, al ejecutar el script de manera local nos devolverá por consola:

```
$ node server.js
Server Runing...
MySQL Connected Successfully...
```


Ahora diseñaremos una serie de eventos a los que el servidor atenderá y que, cuando sucedan, desencadenarán una u otra serie de acciones del lado del cliente, del lado del servidor, o en ambos.

connection

Como su propio nombre indica, este evento salta cada vez que un cliente se conecta al servidor. Como es de esperar, es el evento principal y será padre de todos los demás aunque, para empezar, le daremos la función de mostrar en consola cuando alguien se conecta y de mantener un conteo de conexiones abiertas.

```
io.sockets.on('connection', function(socket){ // Cuando se conecta un socket...
  // Conexion
  connections.push(socket);
  console.log('Connected: %s sockets connected', connections.length); // Unicamente muestra cuantas conexiones hay activas en el momento (por consola)
```

```
Server Runing...
MySQL Connected Successfully...
Connected: 1 sockets connected
```

disconnect

De nuevo, derivado del nombre se ejecuta cuando un cliente cierra la conexión. En este caso su única función es eliminar al usuario que cierra la conexión de la lista de personas conectadas, y resta uno al número de conexiones abiertas en consola.

```
// Desconexion
socket.on('disconnect', function(data){

  users.splice(users.indexOf(socket.username), 1); // Si un usuario cierra su socket, desaparece de la lista de usuarios conectados
  updateUsernames();

  connections.splice(connections.indexOf(socket),1);
  console.log('Disconnected: %s sockets connected', connections.length); // De nuevo, registro de conexiones en consola.
});
```

```
Server Runing...
MySQL Connected Successfully...
Connected: 1 sockets connected
Disconnected: 0 sockets connected
```

new user

Empezamos con uno de los eventos más importantes de nuestra aplicación chat. Este evento se producirá cada vez que un usuario nuevo acceda al chat, no al sitio, al chat. Recogerá su nombre y lo vinculará al socket que se ha abierto al conectarse, para mostrarlo en la lista de usuarios conectados.

Adicionalmente, y añadido con posterioridad, cada vez que un usuario nuevo entra al chat **su cliente únicamente** obtiene los mensajes almacenados en la base de datos.

Esto se ha diseñado así pensando en aumentar la eficiencia del sistema. Con la idea de que hacer una petición a la base de datos por cada cliente cada pocos segundos podría llegar a ser cargante se ha diseñado el script de manera que únicamente se solicita información de la base de datos cuando el usuario no la ha visto en directo, es decir, cuando no estaba conectado.

De esta manera, al entrar un usuario al chat, obtendrá toda la conversación previa existente a través de la base de datos, mientras que los mensajes nuevos serán tratados en tiempo real a través de socket.io. Evidentemente, también se insertan en la base de datos, pero únicamente se solicita la lista completa de mensajes cuando un usuario entra a la sala y, repito, únicamente para él.

```
// Usuario nuevo
socket.on('new user', function(data, callback){
  callback(true);
  socket.username = data;
  users.push(socket.username); // Usuarios conectados

  // Obtener mensajes almacenados cuando el usuario se conecta.
  var query = 'SELECT * FROM webchat_lines ORDER BY id';
  var msgs = con.query(query, function(err, result){
    if(err) throw err;

    socket.emit('load messages', result); // Si la solicitud no devuelve errores, envía todos los mensajes guardados en forma de array al cliente.

    // Testeando salida de datos, para usarla en el chat cliente
    /*for(var x = 0; x < result.length; x++){
      console.log(result[x].name + ' ' + result[x].message);
    }*/
  });

  updateUsernames();
});
```

Como es la primera vez que aparece, lo comento. Algunos de los eventos creados en el servidor contienen una instrucción del tipo “`socket.emit('load messages', result);`” como en la imagen superior. Estas instrucciones envían información al cliente y para **un determinado evento** procesado también en el cliente.

Como veremos más adelante, “`load messages`” será un evento creado en el lado del cliente.

send message

Segundo y último evento base del lado del servidor. Se llevará a cabo cada vez que un usuario presione enter en su teclado, o bien apretando manualmente el botón de enviar, después de haber introducido un mensaje.

Únicamente recoge la información introducida en el mensaje y la vincula a un nombre de usuario, para ingresar el mensaje a la base de datos y, posteriormente, enviar dicho mensaje a todos los clientes. Podría llegar a parecer un proceso de varios segundos pero, llevado a la práctica, es sorprendentemente instantáneo.

```
// Enviar mensaje
socket.on('send message', function(data){
  var query_insert = util.format('INSERT INTO webchat_lines (name, message) VALUES ("%s", "%s")', socket.username, data); // Ingreso del mensaje de texto a la base de datos
  con.query(query_insert, function(err, result){
    if(err) throw err;
    console.log("Mensaje insertado..."); // Si no hay errores, muestra por consola "Mensaje insertado"
  });

  io.sockets.emit('new message', {msg: data, user: socket.username}); // Envío de la información obtenida aquí, al lado del cliente.
});
```

clear messages

Tiene lugar cada vez que un administrador manda “/clear” como contenido de su mensaje en el chat y únicamente borra todos los mensajes de la base de datos para luego ejecutar el evento correspondiente en todos los clientes.

```
// Comando /clear
socket.on('clear messages', function(callback){
  var query_delete = "DELETE FROM webchat_lines"; // Elimina todos los mensajes de la base de datos.
  con.query(query_delete, function(err, result){
    if(err) throw err;
    console.log('Mensajes eliminados...');
    io.sockets.emit('cleared');
  })
});
```

Función updateUserNames()

Para no dejar nada sin explicar, es una función a la que se recurre cada vez que un usuario nuevo se conecta, o se desconecta. Avisa al cliente para que ejecute el evento correspondiente de su lado.

```
function updateUserNames(){
  io.sockets.emit('get users', users);
}
```

Con esto finaliza la creación del script servidor, y me gustaría llegar a una **conclusión final**:

El servidor únicamente establece un par de condiciones y no hace realmente **NADA**, hasta que el cliente lo solicita.

4.2.4.- Cliente

Aunque se trata de un archivo php, bien podría ser únicamente html. La razón de haber escogido la extensión php es meramente funcional, para poder acceder a las variables de sesión que se usan en el resto del sitio web.

Quiero destacar que el lado del cliente podría resultar extraño en cuanto a diseño (ahora veremos por qué) pero se debe únicamente a la ausencia de contenido real en el sitio. Se ha diseñado la sala de chat como si fuera un servicio adicional para la página web, no como algo exclusivo.

Entonces, una vez haber accedido al sitio, se nos pedirá ingresar otro nombre (que puede ser el mismo, o no). Esto se ha hecho pensando en la empresa donde trabajo, con la idea de utilizar cuentas departamentales en lugar de individuales.

Por ejemplo, la cuenta del departamento de administración se llama administración.

Cada empleado del departamento utilizará la misma cuenta para acceder al sitio, pero al acceder al chat, introducirán un nombre alternativo:

Marta (Administración)

Para resumir, el usuario accede al sitio y es redireccionado a la aplicación de chat, donde ingresará un nombre de usuario alternativo y es entonces cuando la sala de chat aparece (está oculta por defecto).

```
#messageArea{  
  display: none; /* La página entera (el contenedor del chat completo), está oculto por defecto*/  
}
```

Introduce tu nombre

Entrar

Usuarios Conectados

Marta (Administracion)

Admin (Admin): Mensajes borrados por un administrador.

Introduce un mensaje

Enviar

Quiero dejar claro antes de seguir, que los diseños gráficos (css) del chat en sí no son míos, a diferencia de los del registro que sí lo son, si no que se ha usado un css prefabricado que se incluirá en la bibliografía.

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">
```

Continuemos, ahora sí, exponiendo la parte lógica dentro del lado del cliente. Evidentemente también programada en JavaScript.

Antes de nada, abriremos el script indicándole al cliente a dónde tiene que conectarse, y definiendo las diferentes secciones del código HTML con las que va a interactuar.

```
<script>
  $(function(){
    var socket = io.connect('http://localhost:3000'); // Conexión al script

    // Definición de zonas del código HTML
    var $messageForm = $('#messageForm');
    var $message = $('#message');
    var $chat = $('#chat');

    var $messageArea = $('#messageArea');
    var $userFormArea = $('#userFormArea');
    var $userForm = $('#userForm');
    var $users = $('#users');
    var $username = $('#username');
```

cada vez que se pulsa enviar

El primer evento del lado del cliente. Cada vez que un usuario intenta enviar un mensaje ya sea a través del botón correspondiente, o presionando enter en su teclado recoge el contenido de la caja de texto y la envía al servidor. Adicionalmente, comprueba si el mensaje coincide con `"/clear"` y si lo hace, avisa al servidor del intento de eliminación de mensajes, y sustituye el mensaje por `"Mensajes eliminados por un Administrador"` (los demás usuarios no llegan a ver `/clear` en chat nunca).

```
$messageForm.submit(function(e){ // Cada vez que se envía un mensaje desde el lado cliente.
  e.preventDefault();
  if($message.val() == '/clear'){
    socket.emit('clear messages');
    $message.val('Mensajes borrados por un administrador.');
  }
  socket.emit('send message', $message.val()); // La información se manda al script del lado del servidor.
  $message.val(''); // Reset
});
```

cada vez que entra un usuario

Cada vez que un usuario envía un nombre para acceder al chat, el cliente recoge éste nombre además del nombre original de la cuenta (a través de una variable de sesión php) y lo envía al servidor.

```
$userForm.submit(function(e){ // Cada vez que entra un usuario nuevo.
    e.preventDefault();
    socket.emit('new user', $username.val() + ' (' + session + ')', function(data){ // Envío de información al script servidor, adjuntando al nombre además la variable de session php.
        // De manera que los usuarios se mostrarán: Nombre-escogido (Nombre real de registro)
        $userFormArea.hide(); // Si se han introducido datos, oculta el formulario de nombre para el chat.
        $messageArea.show(); // Y muestra el chat en sí.
    });
});

$username.val(''); // Reset
});
```

La conversión de la variable de sesión php, para hacerla accesible en javascript, se hace previamente en el código HTML.

```
<script>
    var session = '<?php echo $session_name ?>'; // Conversión de variable de sesion php a js.
</script>
```

load messages

Este es uno de los eventos que se ejecutan a petición del servidor, con información previa del cliente, pero a petición del servidor al fin y al cabo. Cuando entra el usuario, se cargan gráficamente los mensajes obtenidos por el servidor previamente.

```
socket.on('load messages', function(data){ // Recibe información del script servidor para mostrar en pantalla todos los mensajes almacenados al conectarse.
    if(data.length > 0){
        for(var x = 0; x < data.length; x++){ // Bucle que pasa por todas las iteraciones de data (Que viene siendo el resultado del query hecho en el servidor)
            {
                $chat.append('<div class="well"><strong>'+ data[x].name + '</strong>: ' + data[x].message + '</div>');
            }
        }
        scrollDown(); // Baja la barra lateral automáticamente.
    }
});
```

new message

Evento que también se activa a petición del servidor, recopila la información que el servidor a procesado previamente para mostrar en pantalla de los clientes los mensajes que se van enviando, asociados a un nombre y un nombre alternativo.

```
socket.on('new message', function(data){ // Procesa información del servidor para enviar los mensajes nuevos a los clientes.
    $chat.append('<div class="well"><strong>'+data.user+'</strong>: '+data.msg+'</div>');
    scrollDown(); // Baja la barra lateral automáticamente cada vez que se envía un mensaje.
});
```

get users

De nuevo, evento que se ejecuta a petición del servidor, cada vez que un usuario entra o sale del chat recoge la información del servidor y añade/elimina gráficamente cada usuario de la lista.

```
socket.on('get users', function(data){
  var html = '';
  for(i = 0; i < data.length; i++){
    html += '<li class="list-group-item">'+data[i]+'</li>'; // Añade los usuarios que se van conectando a la lista lateral, obteniendo la información del script servidor.
  }
  $users.html(html);
});
```

cleared

Cuando el servidor ya ha eliminado satisfactoriamente todos los mensajes almacenados en la base de datos tiene lugar este evento que, eliminará de igual forma todos los mensajes pero gráficamente.

```
socket.on('cleared', function(callback){
  while(chat.firstChild){
    chat.removeChild(chat.firstChild); // Elimina todos los mensajes mostrados gráficamente, los hijos del contenedor chat.
  }
});1
```

Función scrollDown()

Llegado a determinado número de mensajes en el chat terminarían por desbordar si no fuese porque se trata de un elemento con barra lateral. Entonces, para darle mayor sensación de dinamismo y comodidad al usuario, se ha implementado esta simple función que baja de forma automática la barra lateral cada vez que es llamada.

El usuario puede hacer scroll lateral hacia arriba siempre que quiera para leer conversaciones más antiguas, pero no tendrá la necesidad de descender manualmente la barra cada vez que llegue un mensaje nuevo.

```
function scrollDown(){ // Funcion que únicamente fuerza a la barra de scroll vertical a bajar cada vez que se la llama.
  var elem = document.getElementById('chat');
  elem.scrollTop = elem.scrollHeight;
}
```

Esto concluye la parte del cliente, y como consecuente la aplicación en general. Quisiera añadir un par de anotaciones al respecto:

Los eventos **load messages**, **new message** y **get users** insertan código HTML directamente en las páginas de los clientes.

La aplicación en conjunto no es más que una comunicación continua entre el servidor, el cliente y viceversa. El cliente comienza enviando o solicitando cierta información, y se desencadena toda una conversación entre ambos para llegar a un estado final.

No hay ningún tipo de refresco ni temporizador, la conversación en el chat es absolutamente dinámica.

Usuarios Conectados

Marta (Administracion)

Angel (Desarrollo)

Sabrina (Ventas)

Admin (Admin): Mensajes borrados por un administrador.

Sabrina (Ventas): Buenas tardes!

Angel (Desarrollo): Wow, esta conversación rebosa dinamismo!

Marta (Administracion): Siempre se ha dicho que los de desarrollo sois unos bichos raros...

Introduce un mensaje

Enviar

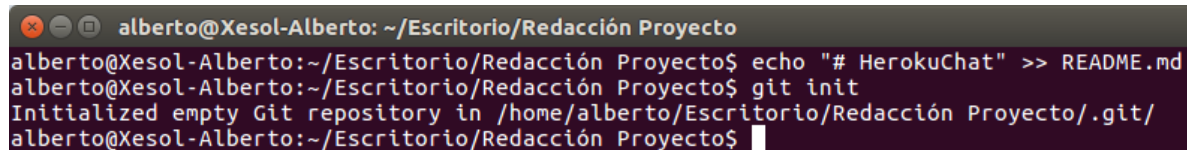
4.3.- Heroku y Github

Como ya he mencionado con anterioridad, mi intención es desplegar la aplicación de chat (la parte servidor `server.js`) en Heroku para mantenerlo en ejecución 24/7. En el momento de escritura de este documento, Heroku ofrece 4000 horas mensuales de ejecución gratuitas si se ha introducido un método de pago válido. Pero, antes de nada, deberemos configurar y subir a la red nuestro propio repositorio git.

4.3.1.- Github

O bien creamos una carpeta separada para la ubicación del repositorio en nuestro equipo, o utilizamos la carpeta existente. Personalmente recomiendo mantener las cosas separadas, en el remoto caso de que algo vaya terriblemente mal tendríamos copia de respaldo tanto en local como en la nube.

Habiendo descargado el paquete git para el sistema operativo bajo el que se opere nos surgen de nuevo dos alternativas, utilizar la consola de comandos para comunicarnos con git, o utilizar una interfaz gráfica preparada para ello como, por ejemplo, [Git Kraken](#). En mi caso utilizaré simplemente la consola bajo Ubuntu 16.14

A terminal window with a dark background and light text. The title bar shows 'alberto@Xesol-Alberto: ~/Escritorio/Redacción Proyecto'. The terminal content shows three lines of commands and their output: 'echo "# HerokuChat" >> README.md', 'git init', and 'Initialized empty Git repository in /home/alberto/Escritorio/Redacción Proyecto/.git/'.

```
alberto@Xesol-Alberto: ~/Escritorio/Redacción Proyecto
alberto@Xesol-Alberto:~/Escritorio/Redacción Proyecto$ echo "# HerokuChat" >> README.md
alberto@Xesol-Alberto:~/Escritorio/Redacción Proyecto$ git init
Initialized empty Git repository in /home/alberto/Escritorio/Redacción Proyecto/.git/
alberto@Xesol-Alberto:~/Escritorio/Redacción Proyecto$
```

A continuación le diremos a Git qué archivos queremos añadir al repositorio. En mi caso concreto, no es necesario exportar todos los archivos de node, únicamente con script.js (que es el servidor en sí) y package.json (detalles del proyecto y dependencias) debería ser suficiente. Heroku procesará el archivo json para instalar aquellas librerías que sean necesarias.

```

alberto@Xesol-Alberto: ~/Escritorio/Redacción Proyecto
alberto@Xesol-Alberto:~/Escritorio/Redacción Proyecto$ echo "# HerokuChat" >> README.md
alberto@Xesol-Alberto:~/Escritorio/Redacción Proyecto$ git init
Initialized empty Git repository in /home/alberto/Escritorio/Redacción Proyecto/.git/
alberto@Xesol-Alberto:~/Escritorio/Redacción Proyecto$ ls
Heroku package.json package-lock.json README.md server.js
alberto@Xesol-Alberto:~/Escritorio/Redacción Proyecto$ git add *
alberto@Xesol-Alberto:~/Escritorio/Redacción Proyecto$ git commit -m "Primer commit"

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'alberto@Xesol-Alberto.(none)')
alberto@Xesol-Alberto:~/Escritorio/Redacción Proyecto$ git config user.email apazgarcia17@outlook.es
alberto@Xesol-Alberto:~/Escritorio/Redacción Proyecto$ git config user.name Kratory
alberto@Xesol-Alberto:~/Escritorio/Redacción Proyecto$

```

Vincularemos ahora el directorio local con el repositorio, por el momento vacío, previamente creado en Github y le diremos a git que suba todos los archivos añadidos.

```

alberto@Xesol-Alberto: ~/Escritorio/Redacción Proyecto
alberto@Xesol-Alberto:~/Escritorio/Redacción Proyecto$ git remote add origin https://github.com/Kratory/HerokuChat.git
alberto@Xesol-Alberto:~/Escritorio/Redacción Proyecto$ git push -u origin master
Username for 'https://github.com': Kratory
Password for 'https://Kratory@github.com':
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (7/7), 9.43 KiB | 0 bytes/s, done.
Total 7 (delta 0), reused 0 (delta 0)
To https://github.com/Kratory/HerokuChat.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
alberto@Xesol-Alberto:~/Escritorio/Redacción Proyecto$

```

Ahora contamos con todos los archivos alojados en Github, disponibles para su edición o descarga en cualquier momento y desde cualquier lugar.

1 commit
1 branch
0 releases
1 contributor

Branch: **master**
New pull request
Create new file
Upload files
Find file
Clone or download

Kratory Frist commit		Latest commit 3c7a726 2 minutes ago
Heroku	Frist commit	2 minutes ago
README.md	Frist commit	2 minutes ago
package-lock.json	Frist commit	2 minutes ago
package.json	Frist commit	2 minutes ago
server.js	Frist commit	2 minutes ago

README.md

HerokuChat

4.3.1.- Heroku

Nos crearemos una cuenta de forma gratuita en [heroku](#) y crearemos una nueva aplicación. El nombre de las aplicaciones en [heroku](#) va siempre en minúsculas y separado por guiones si tuviera espacios.

Una vez escogido el nombre de la aplicación escogeremos la región en la que queremos que la se hospede, y clickaremos en "[create app](#)".


App name

✓

[heroku-chat-proyecto](#) is available

Choose a region


🌐 Europe


 [Add to pipeline...](#)


[Create app](#)


Ahora, en el apartado de [Deploy](#), escogeremos como método de despliegue GitHub, vincularemos nuestra cuenta de [GitHub](#) e introduciremos el nombre del repositorio al que nos queremos conectar.

Deployment method

 [Heroku Git](#)
Use Heroku CLI

 [GitHub](#)
Connect to GitHub

 [Dropbox](#)
Connect to Dropbox

 [Container Registry](#)
Use Heroku CLI


Connect to GitHub

Connect this app to GitHub to enable code diffs and deploys.

Search for a repository to connect to

[Search](#)

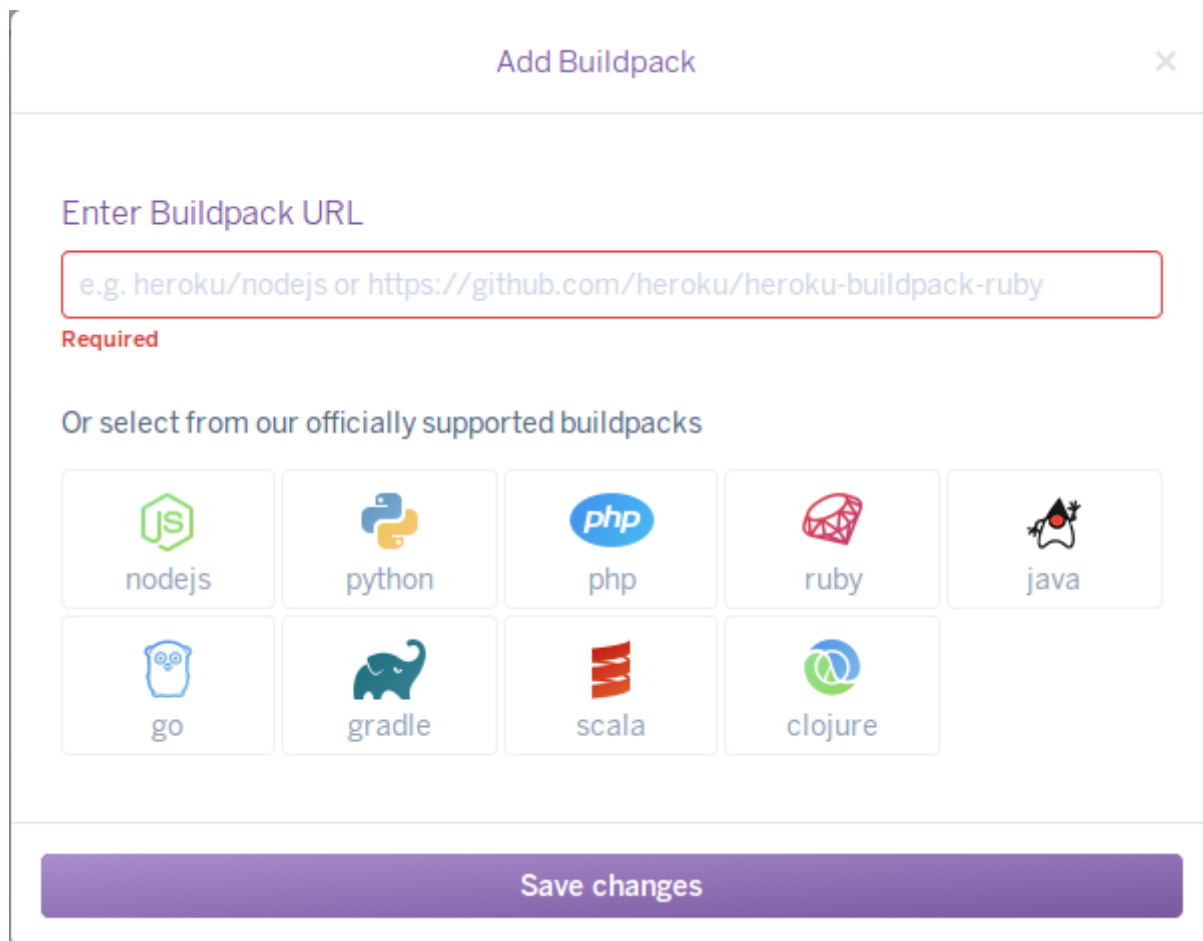
Missing a team? [Ensure Heroku Dashboard has team access.](#)

 Kratory/HerokuChat

[Connect](#)

Si así lo deseamos, podemos activar los despliegues automáticos, de manera que cada vez que actualicemos el repositorio de git, Heroku pare el servicio, lo actualize y lo arranque de nuevo automáticamente.

Iremos ahora a opciones, y añadiremos el buildpack correspondiente al código que queramos ejecutar, en mi caso node.js.



The screenshot shows a modal window titled "Add Buildpack" with a close button (X) in the top right corner. Inside the modal, there is a section labeled "Enter Buildpack URL" with a text input field containing the placeholder text "e.g. heroku/nodejs or https://github.com/heroku/heroku-buildpack-ruby". Below the input field, the word "Required" is written in red. Underneath, there is a section titled "Or select from our officially supported buildpacks" which displays a grid of ten buildpack icons: nodejs, python, php, ruby, java, go, gradle, scala, and clojure. At the bottom of the modal, there is a large purple button labeled "Save changes".

Bajo la categoría **Resources** deberemos activar el dyno correspondiente, que será definido por el Procfile que nosotros hayamos subido, en este caso:

web: node server.js

Para el correcto funcionamiento del script, necesitaremos añadir una base de datos mediante un addOn de Heroku. Este no es un servicio gratuito, motivo por el cual no se ilustrará en este documento.

Sin embargo, tenemos una aplicación hosteada en **Heroku** totalmente funcional (si fuera capaz de acceder a la base de datos, o no contara con una).

5.- Calendario de acciones

Fase de Planificación (20/09/17 ~ 20/10/17)

- ➔ Recopilación de información (30 días)
- ➔ Cambios de idea, confusión en general (25 días)

Fase de desarrollo (21/10/17 ~ 01/11/17)

- ➔ Elaboración de la Aplicación (11 días)
- ➔ Elaboración del entorno PHP (2 días)

Fase final (01/11/17 ~ 03/11/17)

- ➔ Testeo, arreglos y mejoras (3 días)
- ➔ Redacción del proyecto, finalización (3 días)

6.- Evaluación de las acciones del proyecto

Me pasé gran parte del tiempo saltando de una idea a otra, fallando, probando, desechando y empezando de nuevo. Esto, sumado a la necesidad de investigar individualmente cada elemento del proyecto fue lo que me robó más tiempo, aprender a programar en `node`, a usar `socket.io`, `express`, `ajax`... etc.

Luego, por el contrario, una vez establecida una idea clara de proyecto tomó relativamente poco tiempo llegar a un producto decente. Aunque, como siempre, concentré el trabajo en una semana, llegando a trabajar en él 8-10 horas diarias al ver venir la fecha límite.

Durante la fase de testeo de la aplicación, conté con la ayuda de familiares y amigos que se conectarían simultáneamente a mi servidor para comprobar tiempos de respuesta, posibles errores, etc....

7.- Conclusiones finales y líneas futuras

A lo largo del transcurso del proyecto surgieron diversos problemas y complicaciones, pero tengo la satisfacción de poder decir que el producto final se encuentra dentro de las expectativas iniciales, e incluso un poco por encima. No obstante, me hubiera gustado dedicarle más tiempo y haber implementado más funcionalidades como, por ejemplo, mensajería individual (mensajes privados), implementación de emoticonos, detección de diversos comandos como /uinfo (que mostraría información del usuario, como tiempo conectado, nombre completo, etc...), capacidad para insertar imágenes como elementos embed e implementar más de una sala simultánea.

Me molesta enormemente no haber sido capaz de concluir la etapa del proyecto que hace referencia a Heroku, tengo la sensación de que debe de haber alguna manera de hacerlo sin recurrir a métodos de pago, pero por escasez de tiempo y de recursos se va a quedar en el tintero de momento.

A fin de cuentas, el objetivo final del proyecto se ha visto realizado. Aprender y obtener cierto grado de fluidez en otro lenguaje de programación a parte de aquellos que ya estudiamos durante el ciclo.

Tengo intención de seguir expandiendo el proyecto si resulta que me quedo en la empresa, con la idea de llegar a aplicarlo al entorno laboral que es para lo que fue diseñado en primer lugar.

El producto final ahora mismo es una aplicación de chat dinámica, sencilla pero agradable y fácil de usar pero sobretodo funcional, integrada en un entorno php.

8.- Bibliografía

→ PHP

- <http://php.net/manual/es/book.pdo.php>
- <http://php.net/manual/es/intro.pdo.php>
- <http://php.net/manual/es/book.mysql.php>
- <https://stackoverflow.com/questions/41391862/how-to-access-php-session-variable-in-javascript>

→ JAVASCRIPT

- <https://www.w3schools.com/js/>
- <http://expressjs.com/es/guide/routing.html>
- <https://github.com/socketio/socket.io>
- <https://github.com/socketio/socket.io>
- <https://github.com/mysqljs/mysql>

→ CSS

- <https://github.com/twbs/bootstrap>
- <https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css>

→ GITHUB

- <https://github.com/>
- <https://guides.github.com/>

→ HEROKU

- <https://devcenter.heroku.com/>
- <https://devcenter.heroku.com/categories/nodejs>
- <https://devcenter.heroku.com/articles/getting-started-with-nodejs#introduction>

→ MI PROYECTO EN GITHUB

- <https://github.com/Kratory/ProyectoAsir5844>