DOCUMENTACIÓN SUPUESTOS

FIRA VALENCIA

Walter Ponce Garcia - 1º DAW

¿Cuál es nuestro propósito en los supuestos?

Nuestro objetivo es lograr la intención de captar a los ciudadanos fanáticos del ámbito sobre cada supuesto realizado y principalmente, atraer esos ámbitos hacia feria valencia donde podríamos conseguir tener mayor variedad de público, de ahí, mayor demanda en ese sector hacia el mercado cara a los visitantes.

[Supuesto 4: Introducción]

Este el supuesto identificado como supuesto 4, donde, nuestro objetivo es **atraer al público fanático del mundo del motociclismo** hacia nuestras ferias asociada con Feria 2 Ruedas y Feria Valencia. Para ello, realizaremos un correo electrónico llamativo cuya intención es conseguir más datos de los espectadores del mundo del motociclismo para futuros eventos con Feria 2 Ruedas **permitiéndoles participar en un sorteo de entradas al GP de Cheste**.

[Supuesto 4: Análisis]

Bien, ahora que sabemos nuestro propósito con este supuesto, debemos pensar:

1. Tenemos que captar la atención de posibles futuros visitantes conocedores o no previamente a nuestra feria, que principalmente sean aficionados al mundo del motociclismo, lo cual **buscamos nuevos visitantes**.
2. Analizar nuestra posible competencia, en lo cual entra un siguiente análisis:
    a. Debilidades
        i. **Dependencia del premio:** El interés puede estar centrado solo en el sorteo y no en la feria en sí, lo que puede traducirse en registros no cualificados.
        ii. **Limitado alcance temporal**: El plazo hasta el 1 de septiembre reduce el margen para alcanzar una gran audiencia si la campaña no se lanza con suficiente antelación
    b. Amenazas
        i. **Competencia de eventos similares:** Otras ferias o promociones relacionadas con el motociclismo podrían desviar la atención del público objetivo.
        ii. **Percepción de spam:** Si el correo no está bien segmentado o redactado, puede ser percibido como publicidad no deseada y afectar negativamente la imagen de la feria.
    c. Fortalezas:
        i. **Incentivo atractivo:** El sorteo de entradas para un evento popular como el GP de Cheste genera alto interés y puede aumentar significativamente la tasa de respuesta al correo.
        ii. **Segmentación efectiva:** El público objetivo (aficionados al motociclismo) coincide directamente con el perfil de asistentes a la feria, lo que mejora la efectividad del mensaje
    d. Oportunidades:
        i. **Captación de leads y aumento de visibilidad:** La acción permite recopilar datos valiosos de potenciales asistentes interesados, útiles para futuras campañas promocionales. Además, si la promoción es compartida en redes podría llegar a un gran y mayor alcance más allá de la base de datos actual

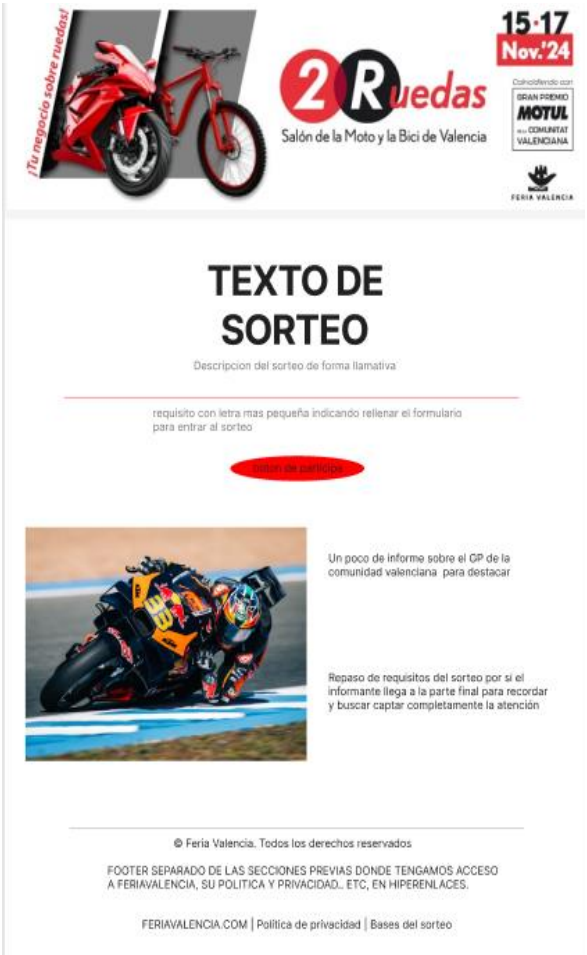
3. Lo más importante, **ampliar nuestros "clientes"** visitantes a nuestra Feria relacionados con el mundo del motociclismo y expandirnos a nuevos ámbitos.

[Supuesto 4: Diseño]

Vamos a desarrollar dos diseños provisionales para hacer idea de cómo queremos que sea la estructura repartida del correo con esa intención de captar la atención a los visitantes.

Puesto a que algunas personas visitan el correo desde escritorio o teléfono, vamos a realizar el modelo en sus dos versiones. Por tanto, aquí van sus maquetas.

Versión escritorio



Versión en teléfono

Hay que tener en cuenta varios detalles, más allá de saber que está enfocado a aficionados del motociclismo:

1. Esto es así para captar visualmente de primera mano al usuario receptor, también para no ver el correo a la primera ojeada y captar toda la información y lo importante, que puede participar en un sorteo gratuito.
2. Hemos usado un botón rojo acorde a colores sobre la cabecera de imagen para tener coherencia y sentido a la vista. No hay muchos separadores, pues con lo insertado sería suficiente para separar una sección de la entrada de información con la descripción sin irnos demasiado lejos de lo importante.
3. Y, por ende, se ha aplicado una jerarquía visual destacando el título principal con tipografía grande a diferencia de la descripción y el texto de los requisitos porque lo importante es decirle al usuario: ¡Oye! Puedes participar en un sorteo sin costo y sobre algo que te gusta. He usado un ajustado espaciado entre secciones según sus niveles de información.

[Supuesto 4: Primeras muestras]

Ahora vamos a pasar a la primera muestra de cómo ha quedado nuestro diseño para el supuesto. Para ello, usaremos el lenguaje **MJML y un entorno de desarrollo como Visual Studio Code. Además, manejaremos un control de versiones que podrán seguir desde nuestro repositorio público,** https://github.com/WPGDAW1/proyecto . También tendremos bases de datos para el control de usuarios.

Diseño en Escritorio:

Diseño completo en teléfono

¡Sorteamos 2 entradas para el GP de Motociclismo de Cheste!

La Feria 2 ruedas y Feria Valencia quieren premiar tu pasión por las motos. Participa en el sorteo y vive la emoción del MotoGP en vivo y en directo.

Solo tienes que rellenar el formulario de inscripción antes del **1 de septiembre** y entrarás en el sorteo de dos entradas doble para el Gran Premio de Cheste. ¡Vamos!

¡Participa ahora!

El Gran Premio de la Comunidad Valenciana en Cheste es una de las últimas carreras del mundial y uno de los eventos más esperados del motociclismo

Fecha del sorteo: 2 de septiembre de 2025
Entradas válidas para el domingo de carrera
Participación gratuita rellenando el formulario.

feriavalencia.com | Política de privacidad | Bases del sorteo

Feria Valencia

[Supuesto 4: Desarrollo y envió de correo - Individual]

He realizado una primera prueba en este momento del proyecto para demostrar si el archivo mjml con lo añadido cumple con lo necesario para aceptar OUTLOOK el correo con imagenes y sus estilos de la siguiente manera:

1. Creamos instancia EC2 donde gestionaremos el servicio SMTP para envio de los correos
2. Aplicamos reglas de salida para permitir el envio de mensajes de correo a través del puerto smtp 587.
3. Antes de nada, enviar por scp a la terminal conectada con la instancia EC2 el archivo mjml con el modelo a usar durante esta prueba
4. Dentro de la instancia EC2, instalar mjml, npm (esto en una linea: **sudo apt install mjml npm –y**).
5. Una vez instalado podemos crear una carpeta especifica donde tengamos todo lo nuevo a instalar a partir de ahora y el archivo mjml
6. Dentro, con npm, instalaremos las dependencias necesarias para operar dentro de la instancia, con javascript + archivo .env para desarrollar lo necesario y enviar el correo al destinatario específico de prueba (n**pm install mjml nodemailer dotenv mysql2**).
7. Asignamos nuestra información para identificar quien envia el correo y quien lo recibe en un .env
8. Creamos el archivo js “send.js” para realizar el envio de correo con titular al destinatario (**este archivo es compatible con correos gmail, outlook, etc**).
9. Para finalizar, ejecutamos **node send.js** y listo!

[Supuesto 4: Desarrollo y envió de correo – grupo de personas]

Ahora trataremos de mostrar una forma de enviar con este desarrollo previo a varias personas específicas y conseguiremos así que esos usuarios receptores no vean el correo de los demás receptores. Luego trabajaremos con el tema de grupo de personas a partir de suscripciones al Newsletter donde entrará csv y base de datos.

Dentro del archivo send.js (LINK REPO)

```
// Opciones del correo
const mailOptions = {
  from: process.env.O365_USER,
  to: process.env.TO,
  bcc: ['correo1', 'correo2', 'correo3']
  subject: 'TITULAR',
  html: htmlOutput
};
```

En la función que gestiona a quien envía el correo, debemos agregar la línea **bcc:** seguido de los correos con la siguiente estructura (formato array en js)

Esto significa **comentar/eliminar la línea TO=correo1, correo2...** del archivo .**env para que no interfiera con el envío de usuarios** especificado directamente en el js.

```
O365_USER=walpongar@alu.edu.gva.es
O365_PASS=Contrasenya_Encriptada
#TO=test-s0b6tgkow@srv1.mail-tester.com
```

Y con esto, nuevamente escribimos node send.js y listo!

La pregunta sería. ¿**Cómo he logrado que el mismo diseño sea compatible enviándolo a un gmail que un outlook**?

En un principio, pensaba que subiendo las imágenes correspondientes al repositorio github en la misma carpeta que el archivo mjml serviría, pues es un repositorio público y sacando enlace https funcionaría uniéndolo a los src de cada imagen en el archivo mjml pero no...

Tuve que conseguir hacer esto **desde un hosting público activo continuamente** que me permitiera **subir las imágenes online "a la nube"**, en este caso, con https://console.cloudinary.com y de ahí, extraer la imagen como si la estuviera queriendo compartir con otro usuario, de ahí **extraje los enlaces** y hasta las imágenes se veían en gmail o outlook sea desde móvil o escritorio y encima adaptado a pantalla.

[Supuesto 4: Desarrollo y envió de correo – grupo de personas listas csv]

Esta parte es la más importante, ya que una vez automatizado, queremos garantizar que envíe el correo correspondiente a cada usuario sin nosotros hacer nada. Para ello, necesitaremos recopilar los datos de todos los suscriptores a la newsletter con por ejemplo un csv (**suscriptores.csv**). Junto a ese archivo, mediante un programa a parte (**enviabbdd.java**) **Realizaremos una combinación de sistema de ficheros** leyendo todas las líneas **a la vez que haremos gestión de base de datos** conectando y enviando mediante "**insert**" cada registro con su correspondiente información.

Muestra del código para mostrar su funcionamiento

```java
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class enviabbdd {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {

        try {
            Connection con = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/FeriaValencia", "Admin", "Admin");
            File f = new File("suscriptores.csv");
            FileReader fr = new FileReader(f);
            BufferedReader br = new BufferedReader(fr);
            String linea = br.readLine();
            int count = 0;

            while(linea != null){

                String [] datos = linea.split(",");
                String nombre = datos[0];
                String email = datos[1];
                String idioma = datos[2];
                int suscrito = Integer.parseInt(datos[3]);

                String sql = "INSERT INTO suscriptores (nombre, email, idioma, suscrito) VALUES (?,?,?,?)";
                PreparedStatement st = con.prepareStatement(sql);
                st.setString(1, nombre);
                st.setString(2, email);
                st.setString(3, idioma);
                st.setInt(4, suscrito);
                st.executeUpdate();
                count++;
                linea = br.readLine();

            }
            System.out.println("Se agregaron "+ count + " registros correctamente.");
        } catch (Exception e) {
            System.out.println("Excepción: "+e.toString());
        }
    }
}
```

Con este programa, mientras el archivo se llame "**suscriptores.csv**", se encuentre en la misma carpeta que el propio programa y las credenciales a la base de datos otorgada sea la correcta, funcionará sin problemas.

Bien, ahora que tenemos en la base de datos todos los registros recibidos del csv y podemos operar, necesitamos actualizar nuestro archivo js que creamos previamente para enviar correos a cada usuario, pero en vez de recibirlo en el **TO=** del archivo .env que teníamos, se hará la búsqueda dentro del nuevo archivo **senddesdebd.js o sendyregistro.js** (este segundo envia el correo y registra el correo en la bbdd) que son modelos actualizados del **send.js.**

Para esto hay cambios:

1. Actualización en el archivo .env agregando credenciales de usuario y base de datos mysql

```
DB_HOST=host
DB_USER=Usuario
DB_PASS=ContraUsuario
DB_NAME=Database
```

2. Creamos conexión y hacemos la consulta necesaria para tener volcado los usuarios que buscamos en la base de datos

```
(async () => {
const connection = await mysql.createConnection({
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
    password: process.env.DB_PASS,
    database: process.env.DB_NAME
});
const [usuarios] = await connection.execute('SELECT nombre,email,idioma FROM suscriptores WHERE suscrito = 1');
for (const user of usuarios){
    const lang = user.idioma;
    let mjmlPath = `oficial_${lang}.mjml`;

if(!fs.existsSync(mjmlPath)){
    console.warn(`Plantilla no encontrada para idioma ${lang}, usando español por defecto.`);
    mjmlPath = 'oficial.mjml';
}

// leemos el archivo
let mjmlTemplate = fs.readFileSync(mjmlPath, 'utf8');
```

3. Puesto a que es una actualización al código de **send.js**, deberemos cambiar a que usuario envía el correo.

```
const mailOptions = {
  from: process.env.O365_USER,
  to: user.email,
  subject: subject,
  html: htmlOutput
};
```

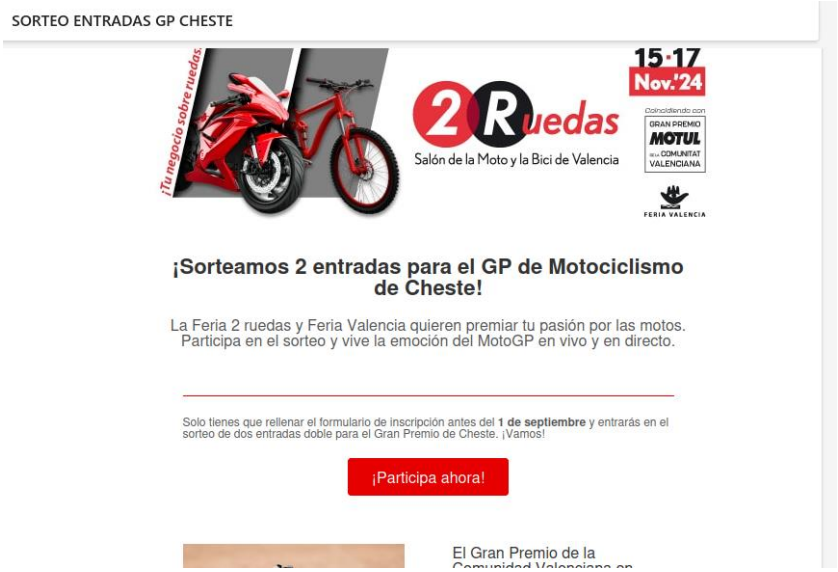
Comprobamos entonces que las credenciales que pusimos en .env sobre nuestro usuario en la base de datos y la base de datos coincidan y luego hacemos la prueba de envío sobre este archivo js.

Correo enviado usando el nuevo archivo js que vuelca el correo de suscriptores y envía el modelo correspondiente a su idioma **“es, en o val”**

```
Correo enviado a walpongar@alu.edu.gva.es: 250 2.0.0 OK <a59d1a14-e21a-8e79-7acc-3085aac2528f@alu.edu.gva.es> [
05.EURP194.PROD.OUTLOOK.COM]
Correo enviado a walter.28.ponce@gmail.com: 250 2.0.0 OK <762f8648-7370-cd8c-31b0-3af42e90bfe5@alu.edu.gva.es>
305.EURP194.PROD.OUTLOOK.COM]
Correo enviado a walter.garcia28@hotmail.com: 250 2.0.0 OK <d3f0c4c0-34e4-9a92-5f3e-e4391d7a1604@alu.edu.gva.es
B0305.EURP194.PROD.OUTLOOK.COM]
```

Una vez comprobado que ha funcionado correctamente, solo actualizamos en crontab, nuestro programador de tareas que automatiza el envío del correo, a, en vez de usar **send.js, usar el nuevo archivo actualizado senddesdebd.js.**

Ejemplo Correo recibido usuario idioma español en escritorio:



Ejemplo correo recibido idioma inglés desde teléfono y correo gmail.

2 Ruedas
Salón de la Moto y la Bici de Valencia

15·17 Nov.'24

Coincidiendo con
GRAN PREMIO MOTUL de la COMUNITAT VALENCIANA

FERIA VALENCIA

We're giving away 2 tickets to the Cheste MotoGP!

The 2 Wheels Fair and Feria Valencia want to reward your passion for motorcycles. Enter the giveaway and experience the thrill of MotoGP live.

Just fill in the registration form before **September 1st** and you'll be entered into the draw for two double tickets to the Cheste Grand Prix. Let's go!

Enter now!

Antes de pasar al siguiente punto, explicaré mi código para entenderlo completamente:

```
const fs = require('fs');
const mjml = require('mjml');
const nodemailer = require('nodemailer');
const mysql = require('mysql2/promise');
require('dotenv').config();
```

En esta primera parte, el programa carga las librerías de las dependencias necesarias para el envío de correo con node.js, además de cargar el framework mjml para cargar el diseño de correo a enviar.

```
const transporter = nodemailer.createTransport({
  host: 'smtp.office365.com',
  port: 587,
  secure: false,
  auth: {
    user: process.env.O365_USER,
    pass: process.env.O365_PASS,
  },
  tls: {
    ciphers: 'SSLv3'
  }
});
```

A continuación, creamos una función de la librería nodemailer para gesitionar el envío de correo, desde que servidor host se provee el envío de correo, bajo que puerto, y que usuario enviará dicho correo, aquí participa el previo archivo .env que recopila las credenciales de usuario y este archivo js con el codigo **process.env.variable** recopila la información del usuario.

```
(async () => {
  const connection = await mysql.createConnection({
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
    password: process.env.DB_PASS,
    database: process.env.DB_NAME
  });
```

La siguiente parte realiza la carga de una promesa con la libreria mysql2/promise. Una promesa (entre ellas async/await), es como un objeto que permite manejar tareas asincrónicas sobre la base de datos, permitiendo realizar una ejecución en el código y definir que realizar cuando tenga algún resultado

```
let exp = "expositor";
const [usuarios] = await connection.execute(
  'SELECT idUsuario, nombre, email, idioma FROM suscriptores WHERE suscrito = 1 AND tipoUsuario = ?',[exp]
);

const subjects = {
  es: "OPTIMIZA TU PARTICIPACIÓN EN EXPOJOVE",
  en: "OPTIMIZE YOUIR PARTICIPATION IN EXPOJOVE",
  val: "OPTIMITZA LA TEUA PARTICIPACIÓ EN EXPOJOVE",
};

for (const user of usuarios) {
  const lang = user.idioma;
  const subject = subjects[lang] || subjects['es'];
  const fechaEnvio = new Date();

  let mjmlPath = `oficial2_${lang}.mjml`;
  if (!fs.existsSync(mjmlPath)) {
    console.warn(`Plantilla no encontrada para idioma ${lang}, usando español por defecto.`);
    mjmlPath = 'oficial2_es.mjml';
  }

  let mjmlTemplate = fs.readFileSync(mjmlPath, 'utf8');
  mjmlTemplate = mjmlTemplate.replace('{{nombre}}', user.nombre);
  mjmlTemplate = mjmlTemplate.replace('{{email}}', user.email);
  const htmlOutput = mjml(mjmlTemplate).html;
```

Seguido a ello, viene la participación con la base de datos:

1. La parte superior realiza una consulta con la base de datos para encontrar a todos los usuarios que cumplan la condiciones y los almacena en un array de usuarios para luego volcar la información de cada uno.
2. La parte de la variable subjects se especifíca para luego según el prefijo de idioma de usuario escoger un asunto concreto a enviar.
3. Cuando tenemos volcado sobre el array a todos los usuarios, empezamos a recorrer a cada usuario en un foreach (**const user of usuarios**) de usuarios para comprobar datos antes de enviar el correo. Dentro, tenemos variables que según el idioma establecerán el correo a enviar según la preferencia indicada. Busca el archivo en base al idioma y si carga, traduce el archivo a html para que sea información legible en el correo.

```
let id_mensaje = null;
try {
  const [mensajeRows] = await connection.execute(
    'SELECT idMensaje FROM mensaje WHERE nombre = ?',
    [subject]
  );

  if (mensajeRows.length === 0) {
    const [insertResult] = await connection.execute(
      'INSERT INTO mensaje (nombre, tipo) VALUES (?, ?)',
      [subject, 'exposición']
    );
    idMensaje = insertResult.insertId;
    console.log(`Mensaje insertado con ID ${id_mensaje}`);
  } else {
    idMensaje = mensajeRows[0].idMensaje;
  }
} catch (err) {
  console.error('Error al procesar la tabla mensaje:', err);
  continue;
}

if (!user.idUsuario || !idMensaje || !fechaEnvio) {
  console.error('Datos incompletos para registrar envío:', {
    idUsuario: user.idUsuario,
    idMensaje,
    fechaEnvio
  });
  continue;
}
```

En esta parte, comprobamos en la base de datos el tipo de mensajes que queremos entregar al usuario y, si no existiera previamente, lo asigna con su información necesaria. En caso de fallas, muestra por terminal el error y motivo por el que no se haya podido asignar el tipo de mensaje en la base de datos.

```
const mailOptions = {
  from: process.env.O365_USER,
  to: user.email,
  subject: subject,
  html: htmlOutput
};
```

Antes de pasar al final, creamos una variable que recopile qué usuario envia el correo, a que correo de cada usuario receptor, el asunto en base al idioma de usuario e igual con el modelo del correo en base al idioma del usuario cargado previamente.

```
    try {
      const info = await transporter.sendMail(mailOptions);
      console.log(`Correo enviado a ${user.email}:`, info.response);

      await connection.execute(
        'INSERT INTO envios (idUsuario, idMensaje, fecha_envio) VALUES (?, ?, ?)',
        [user.idUsuario, idMensaje, fechaEnvio]
      );
    } catch (err) {
      console.error(`Error al enviar a ${user.email}:`, err);
    }
  }

  await connection.end();
})();
```

Para terminar, nos encontramos con la gestión del envío del correo. Hay que recordar que, **el objeto async realiza la ejecución de código y luego esperará para devolver un resultado en función de lo ocurrido con la ejecución.** Bien, esta es la parte que realiza la ejecución final y devolverá ese dicho resultado.

Por un lado, carga la función creada de la variable transporter, donde creamos la función para el envío de correo y quién lo va a enviar y utiliza una función de la libreria cargada con nodemailer para envíar el correo a cada usuario especificado en la variable “mailOptions” (**la anterior imagen muestra esa información**). Una vez realizado el envío, primero comprueba si se ha enviado correctamente el correo, de lo contrario, muestra los errores indicando su motivo (**normalmente puede ocurrir por asignación erronea de credenciales de usuario, o especificación incorrecta en la base de datos asignada en el archivo .env**).

También puede ocurrir que envíe el correo, pero no lo registre, en ese caso interfiere directamente problemas dentro del mismo archivo js relacionados con la carga de la base de datos, incorrectas variables especificadas o incluso problema con los campos a insertar datos en la tabla de envios porque no se ha escrito bien algún campo.

[Supuesto 4: Desarrollo y envió de correo – Automatización del correo con crontab]

Una vez diseñado todo correctamente y verificado que funciona, he asignado a **crontab**, el fichero encargado de poder gestionar tareas programadas en linux **(en este caso linux porque es el sistema operativo de la instancia que he utilizado para este**

proyecto), lo necesario para estar gestionando en todo momento el envio del archivo send.js sin depender de que yo lo envíe. Esto se hace con

Crontab –e o sudo crontab –e

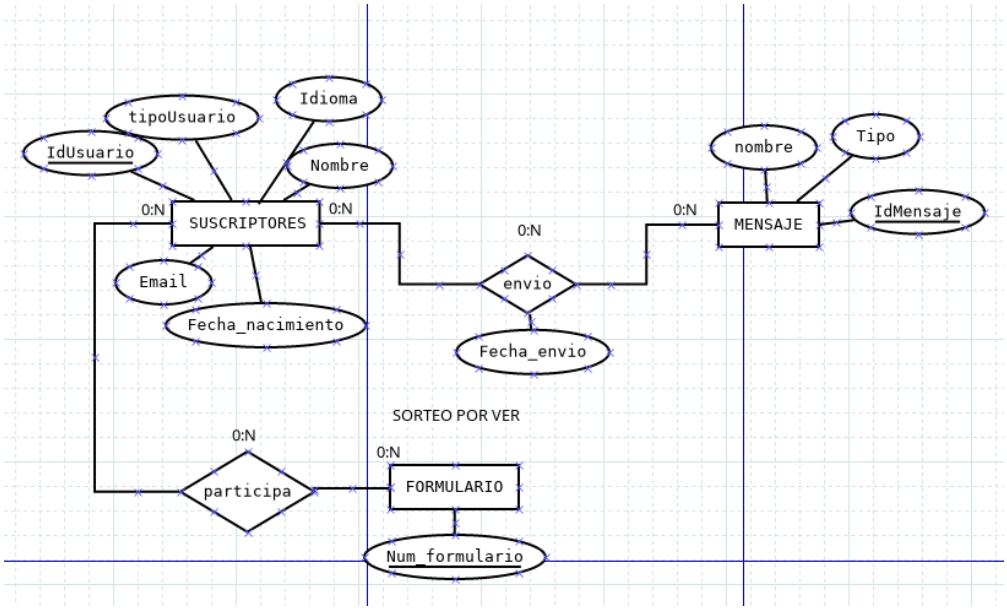
Dentro de ahí asigno una tarea programada todos los días a las 16 para enviar el correo entre las personas asignadas en el js. Puesto a que tengo un archivo js dependiente del .env para saber las credenciales de usuario. En la línea de la tarea accederé con cd a la carpeta y de ahí ejecutaré el archivo para poder hacer uso del .env sin que interfiera crontab.

Esto se hace debido a que crontab opera las tareas programadas desde $HOME.

[Supuesto 4: Gestión de datos y registro de envíos de correo]

Anteriormente, cuando comentaba sobre el programa que a través de los usuarios existentes en la base de datos obtuviera los email que estaban suscritos, previamente teníamos una base de datos creada donde tenemos un sistema de información de datos almacenados para una vez ejecutando el archivo js, estos no solo envíen correos si no que luego nosotros por internos lugares sepamos cuando se ha enviado el correo, de que titular enviamos el correo a dicho usuario y a que usuario final llego ese correo.

[Supuesto 4: Gestión de datos - Modelo E: R]



Nos centraremos en la parte superior, la parte de" Formulario" y el texto" Sorteo por ver" son propuestas extras que comentaremos al final.

Tenemos:

1. Entidad suscriptores que gestiona la información y datos del usuario suscriptor quién recibirá el correo electrónico.
2. Entidad mensajes quién gestiona luego al registrar el correo enviado que tipo de mensaje ha recibido dicho usuario en el correo electrónico.

Por el resto, hablaremos más adelante. Lo importante es que, con esta estructura, podremos saber a qué usuario se está enfocando el correo independientemente sea "**expositor**" como si es un "**visitante**", entre otros ejemplos.

[Supuesto 4: Gestión de datos - Normalización y paso a tablas]

Para asegurarnos de que vamos a estructurar el diseño de la base de datos de manera adecuada transformándolo a tablas, vamos a analizar su normalización.

1: N – Para esta ocasión, **hemos enfocado la base de datos** entendiendo que, **si un usuario es suscriptor de una newsletter** independiente sea un visitante interesado de noticias, promociones, sorteos... etc, como si es expositor, comerciante, entre otros, **realizará dichas suscripciones a las newsletters con un correo electrónico** personal o de trabajo, **pero solo con una de ellas**. Por lo tanto, no hay que actualizar nada.

Por el resto, **2: N y 3: N**, todas las tablas cumplen con las normalizaciones.

Una vez analizado su proceso de normalización veamos cómo queda su modelo lógico:

suscriptores {idUsuario, nombre, email, idioma, suscrito, fecha_nac, tipoUsuario}

VNN: nombre, email, idioma, suscrito, fecha_nac, tipoUsuario

UQ: email

PK: idUsuario

mensaje {idMensaje, nombre, tipo}

PK: idMensaje

envios{id, idUsuario, idMensaje, fecha_envio}

VNN: idUsuario, idMensaje, fecha_envio

PK: id

FK: idUsuario > suscriptores.idUsuario

FK: idMensaje > mensaje.idMensaje

Hemos creado una nueva tabla entidad llamada envios que tiene que ver previamente con la relación **envía** porque al paso del modelo relacional solo puede representarse como una tabla adicional, **ya que las relaciones no pueden tener atributos directamente**. Entonces se creó la tabla con identificador las claves foráneas en relación con suscriptores y mensaje y el atributo fecha_envio como columna de la tabla.

Una vez comprobado, realizamos el paso a tablas. Siguiendo este enlace, **accederás al repositorio donde contiene el archivo SQL** con toda la estructura **principal** creada sobre esta base de datos **(la tabla prevista de formulario y sorteo la comentaremos más tarde**).

[Supuesto 4: Detalles adicionales: Idioma de correo en base a idioma de usuario]

Dato secundario, pero de agradecer para atraer a una completa audiencia y no solo audiencia local, hemos conseguido asignar variables en el archivo js para que, en función del idioma del usuario podamos entregarle el mismo correo con el mismo propósito, pero en su idioma para que sea legible e igual de entendible.

A lo cual entra aquí los nuevos modelos de archivo **oficial_es.mjml (**básicamente oficial_mjml pero para entenderlo el código js), **oficial_es.mjml y oficial_val.mjml.**

Creamos una variable constante que busque coincidencia con el identificador del idioma de usuario y si coincide con alguna de estas enviará al correo electrónico el correo y titular con el idioma correspondiente. Muestra de la parte exacta del código en el archivo js:

```
const [usuarios] = await connection.execute('SELECT nombre,email,idioma FROM suscriptores WHERE suscrito = 1');
for (const user of usuarios){
        const lang = user.idioma;
        let mjmlPath = `oficial_${lang}.mjml`;

if(!fs.existsSync(mjmlPath)){
        console.warn(`Plantilla no encontrada para idioma ${lang}, usando español por defecto.`);
        mjmlPath = 'oficial.mjml';
}

// leemos el archivo
let mjmlTemplate = fs.readFileSync(mjmlPath, 'utf8');
// Remplazando variables
mjmlTemplate = mjmlTemplate.replace('{{nombre}}', user.nombre);

const htmlOutput = mjml(mjmlTemplate).html;
const subjects = {
        es: "SORTEO ENTRADAS GP CHESTE",
        en: "GP CHESTE ENTRIES GIVEAWAY",
        val: "SORTEIG ENTRADES GP XEST",
};
const subject = subjects[lang]
```

Para confirmar que también el titular principal del correo se envíe con el idioma correcto, tenemos la variable constante ”subjects” que como dije, según que idioma coincida con la del usuario también actualizara el titular con el idioma correcto. Una vez tenemos la variable la asignamos en el envío de correo:

```
const mailOptions = {
  from: process.env.O365_USER,
  to: user.email,
  subject: subject,
  html: htmlOutput
};
```

[Supuesto 3: Introducción]

Nuestro objetivo es breve. A diferencia del anterior supuesto donde entran factores de promoción y sorteo de manera interna con la información a captar en el correo electrónico desarrollado, en este caso **debemos realizar un correo informativo** para, una vez que sabemos que nuestro expositor de Expo Jove ya ha acordado con Feria Valencia para su siguiente exposición, **comentarle y ofrecerle todos los servicios disponibles en el correo** y darle redirección de forma fácil si está interesado en ampliar los servicios necesarios para la exposición. Tendremos que informar todo en el correo, pero visualmente necesitaremos que sea llamativo y de interés a primera vista al expositor para echar un vistazo a todo lo que ofrece Feria Valencia.

[Supuesto 3: Análisis]

Bien, ahora que sabemos nuestro propósito con este supuesto, debemos pensar:

1. Como en este supuesto ya hay acordado la exposición con Expo Jove, hay que intentar incentivar su expansión en la exposición si creemos que puede ser necesario nuestros servicios. Por tanto, deberemos tener en cuenta:
2. Analizar nuestra competencia con servicios de otros eventos, en lo cual entra un siguiente análisis:
    a. Debilidades
        i. **Posible baja tasa de apertura** si el asunto o diseño del correo no son lo suficientemente atractivos.
        ii. **Desconocimiento o confusión sobre los servicios disponibles**, si no se presentan de forma clara y estructurada.
    b. Amenazas
        i. **Competencia de otros eventos** que puedan ofrecer servicios similares con mejor comunicación.
        ii. **Falta de respuesta o acción** si el proceso de solicitud de servicios es percibido como complejo o poco accesible.
    c. Fortalezas:
        i. **Base de datos consolidada de expositores anteriores**, lo que permite una comunicación directa y segmentada.
        ii. **Reconocimiento de marca**: Expo Jove es un evento conocido, lo que da legitimidad y atención al mensaje.
    d. Oportunidades:
        i. **Incrementar la satisfacción y fidelización de expositores** al recordarles los beneficios incluidos.
        ii. **Promover servicios adicionales** como patrocinios, mejoras de visibilidad o espacios premium.

3. Lo más importante, **salir a beneficio por partes iguales**, conseguir captar la oferta de servicios si creemos que pueda ser una buena idea para atraer más gente, entre otras cosas, a Feria Valencia.

[Supuesto 3: Diseño]

Vamos a desarrollar dos diseños provisionales para hacer idea de cómo queremos que sea la estructura repartida del correo con esa intención de mostrar los servicios a los expositores.

Puesto a que algunas personas visitan el correo desde escritorio o teléfono, vamos a realizar el modelo en sus dos versiones. Por tanto, aquí van sus maquetas.

Versión escritorio:

TEXTO DE BIENVENIDA, CON VARIABLE A REMPLAZAR POR SU NOMBRE

Aquí damos saludo al expositor/a y le indicamos por su estuviera interesado/a de alguno de los servicios que podemos ofrecerle

ponemos todos los servicios. Buscar ordenarlos y verse limpio

BOTONERA GRANDE PARA REDIRECCIONAR A LA ZONA DE LOS SERVICIOS

Recordatorio de acceso con sus credenciales a la página con centralservicios. Variable para indicar el correo receptor aquí

Versión teléfono:

832 × 381

TEXTO DE BIENVENIDA, CON VARIABLE
A REMPLAZAR POR SU NOMBRE

Aquí damos saludo al expositor/a y le indicamos por su estuviera interesado/a de alguno de los servicios que podemos ofrecerle

ponemos todos los servicios. Buscar ordenarlos y verse limpio. Aquí habran mas separadores

BOTONERA GRANDE PARA REDIRECCIONAR A LA ZONA DE LOS SERVICIOS

Recordatorio de acceso con sus credenciales a la página con centralservicios.
Variable para indicar el correo receptor aquí

[Supuesto 3: Primeras muestras]

Ahora vamos a pasar a la primera muestra de cómo ha quedado nuestro diseño para el supuesto. Para ello, usaremos el lenguaje **MJML y un entorno de desarrollo como Visual Studio Code. Además, manejaremos un control de versiones que podrán seguir desde nuestro repositorio público,** https://github.com/WPGDAW1/proyecto/modelo-oficial-2/ . También tendremos bases de datos para el control de usuarios.

Diseño en escritorio:

Diseño en teléfono:

¡Bienvenido/a a Expojove 2024-2025!

Estimado/a {{nombre}},

Nos complace contar con tu participación en la próxima edición de Expojove, que se celebrará del 26 de diciembre de 2024 al 4 de enero de 2025 en Feria Valencia.

Para facilitar tu preparación y maximizar tu presencia en el evento, ponemos a tu disposición una variedad de servicios que puedes solicitar a través de nuestro portal de expositor.

🛠️ Infraestructura técnica

- Suministro eléctrico para iluminación y equipos
- Conexión a Internet/Wi-Fi
- Toma de agua para actividades específicas

👷 Servicios logísticos

- Almacenaje de materiales
- Transporte interno de mercancía
- Asistencia para montaje y desmontaje

🍽️ Catering y hospitalidad

- Servicio de catering o vending en el stand
- Reserva de restaurantes o zonas VIP

🧽 Limpieza y mantenimiento

- Limpieza diaria del stand
- Gestión de residuos

📢 Publicidad y promoción

- Inserción en el catálogo oficial
- Publicidad dentro del recinto (pantallas, vinilos)
- Rótulo personalizado para el stand

🧑‍💼 Recursos humanos

- Azafatas/os para atención al público
- Intérpretes para expositores internacionales
- Apoyo en control de acceso o seguridad privada

🎪 Montaje de stand y decoración

- Stand modular estándar
- Decoración personalizada y vinilos
- Tarima y moqueta de color
- Alquiler de mobiliario adicional

🚨 Servicios complementarios

- Asistencia sanitaria (botiquín)
- Espacios para reuniones privadas
- Asistencia técnica durante el evento

Solicitar servicios en el Portal del Expositor

Recuerda acceder con tus credenciales. Para asistencia, contacta con centraldeservicios@feriavalencia.com. Correo enviado a {{email}}.

[Supuesto 3: Desarrollo y envió de correo – grupo de personas listas csv]

Esta parte es la más importante, ya que una vez automatizado, queremos garantizar que envíe el correo correspondiente a cada usuario sin nosotros hacer nada. Para ello, necesitaremos recopilar los datos de todos los suscriptores a la newsletter con por ejemplo un csv (**expositores.csv**). Junto a ese archivo, mediante un programa a parte (**enviabbdd2.java**) **Realizaremos una combinación de sistema de ficheros** leyendo todas las líneas **a la vez que haremos gestión de base de datos** conectando y enviando mediante "**insert**" cada registro con su correspondiente información.

Muestra del código para mostrar su funcionamiento

```java
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class enviabbdd {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {

        try {
            Connection con = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/FeriaValencia", "Admin", "Admin");
            File f = new File("suscriptores.csv");
            FileReader fr = new FileReader(f);
            BufferedReader br = new BufferedReader(fr);
            String linea = br.readLine();
            int count = 0;

            while(linea != null){

                String [] datos = linea.split(",");
                String nombre = datos[0];
                String email = datos[1];
                String idioma = datos[2];
                int suscrito = Integer.parseInt(datos[3]);

                String sql = "INSERT INTO suscriptores (nombre, email, idioma, suscrito) VALUES (?,?,?,?)";
                PreparedStatement st = con.prepareStatement(sql);
                st.setString(1, nombre);
                st.setString(2, email);
                st.setString(3, idioma);
                st.setInt(4, suscrito);
                st.executeUpdate();
                count++;
                linea = br.readLine();

            }
            System.out.println("Se agregaron "+ count + " registros correctamente.");
        } catch (Exception e) {
            System.out.println("Excepción: "+e.toString());
        }
    }
}
```

Con este programa, mientras el archivo se llame "**expositores.csv**", se encuentre en la misma carpeta que el propio programa y las credenciales a la base de datos otorgada sea la correcta, funcionará sin problemas.

Bien, ahora que tenemos en la base de datos todos los registros recibidos del csv y podemos operar, necesitamos actualizar nuestro archivo js que creamos previamente para enviar correos a cada usuario, pero en vez de recibirlo en el **TO=** del archivo .env que teníamos, se hará la búsqueda dentro del nuevo archivo **sendyregistro2.js** (este segundo envia el correo y registra el correo en la bbdd) que son modelos actualizados del **send.js.**

Para esto hay cambios:

4. Actualización en el archivo .env agregando credenciales de usuario y base de datos mysql

```
DB_HOST=host
DB_USER=Usuario
DB_PASS=ContraUsuario
DB_NAME=Database
```

5. Creamos conexión y hacemos la consulta necesaria para tener volcado los usuarios que buscamos en la base de datos

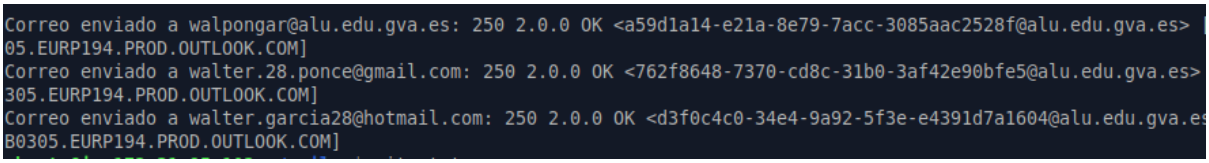
```
(async () => {
const connection = await mysql.createConnection({
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
    password: process.env.DB_PASS,
    database: process.env.DB_NAME
});
const [usuarios] = await connection.execute('SELECT nombre,email,idioma FROM suscriptores WHERE suscrito = 1');
for (const user of usuarios){
    const lang = user.idioma;
    let mjmlPath = `oficial_${lang}.mjml`;

if(!fs.existsSync(mjmlPath)){
    console.warn(`Plantilla no encontrada para idioma ${lang}, usando español por defecto.`);
    mjmlPath = 'oficial.mjml';
}

// leemos el archivo
let mjmlTemplate = fs.readFileSync(mjmlPath, 'utf8');
```

6. Puesto a que es una actualización al código de **send.js**, deberemos cambiar a que usuario envía el correo.

```
const mailOptions = {
  from: process.env.O365_USER,
  to: user.email,
  subject: subject,
  html: htmlOutput
};
```

Comprobamos entonces que las credenciales que pusimos en .env sobre nuestro usuario en la base de datos y la base de datos coincidan y luego hacemos la prueba de envío sobre este archivo js.

Correo enviado usando el nuevo archivo js que vuelca el correo de suscriptores y envía el modelo correspondiente a su idioma **“es, en o val”**



Una vez comprobado que ha funcionado correctamente, solo actualizamos en crontab, nuestro programador de tareas que automatiza el envío del correo, a, en vez de usar **send.js, usar el nuevo archivo actualizado senddesdebd.js.**

Ejemplo Correo desde gmail / Outlook recibido usuario idioma español en teléfono:

OPTIMIZA TU PARTICIPACIÓN EN EXPOJOVE

¡Bienvenido/a a Expojove 2024-2025!

Estimado/a Paco,

Nos complace contar con tu participación en la próxima edición de Expojove, que se celebrará del 26 de diciembre de 2024 al 4 de enero de 2025 en Feria Valencia.

Para facilitar tu preparación y maximizar tu presencia en el evento, ponemos a tu disposición una variedad de servicios que puedes solicitar a través de nuestro portal de expositor.

🛠️ Infraestructura técnica

[Supuesto 3: Gestión de datos]

En este punto no encontramos novedades porque el desarrollo funcionó por igual con la misma base de datos tanto para el supuesto 4 como este supuesto 3.

Para echar un vistazo, ves al punto "Supuesto 4: Gestión de datos y registro de envíos de correos" (CTRL + CLIC).

[Supuesto 3: Detalles adicionales: Idioma de correo en base a idioma de usuario, especificación de nombre y más]

Para este supuesto, el cual es más informativo a expositores, he querido ser más "detallado".

Si bien en el anterior supuesto solo especifiqué variables internas en conexión con la base de datos para que, en función del idioma preferido de usuario, enviara el modelo de correo traducido a su idioma (sea español, valenciano u inglés), he agregado variables de nombre y correo para indicar su nombre en el saludo, añadiendo un toque particular y en la parte inferior especificando para indicar que este correo es el mismo que a quién llego ese mismo mensaje.

¿Qué ha cambiado dentro del archivo de envío de correos?

Simplemente, 2 cosas:

1. Remplazo en el mjml de la variable que llama a nombre
2. Remplazo en el mjml de la variable que llama al correo

A continuación, adjuntamos captura de la parte modificada en el archivo:

```
let mjmlTemplate = fs.readFileSync(mjmlPath, 'utf8');
mjmlTemplate = mjmlTemplate.replace('{{nombre}}', user.nombre);
mjmlTemplate = mjmlTemplate.replace('{{email}}', user.email);
const htmlOutput = mjml(mjmlTemplate).html;
```