Apuntes SWEB:
CSS: https://www.w3schools.com/css/default.asp o bootstrap
https://developer.mozilla.org/es/docs/Web/HTTP/Cookies
(registro) https://www.freecodecamp.org/espanol/news/como-construir-una-forma-de-registro-con-etiquetas-flotantes-y-transiciones-usando-solamente/
(localstorage) https://www.freecodecamp.org/espanol/news/como-usar-localstorage-en-javascript/
(cookies) https://expressjs.com/en/starter/examples.html

https://www.digitalocean.com/community/tutorials?q=login
https://www.geeksforgeeks.org/nodejs/?ref=outind

Pasos para resolver el Apartado 1:

De momento no tenemos del todo claro cuál va a ser el nombre definitivo de la tienda y ahora mismo lo tenemos **hardcodeado en cada ruta**. Si quisiéramos cambiar el nombre tendríamos que modificar esos archivos y cualquier futura ruta que creemos. **Modifique la aplicación para definir en un único sitio el nombre de la tienda** (ahora mismo "Embutidos León"), que este sea accesible en las vistas y elimine el nombre que aparece ahora en las rutas.

https://www.geeksforgeeks.org/how-to-create-global-variables-accessible-in-all-views-using-express-nodejs/

1. **Crear un archivo de configuración:**
   - Crea un archivo llamado `config.js` en la raíz del proyecto.
   - Define el nombre de la tienda en este archivo:

```
module.exports = {
  shopName: "Embutidos León",
};
```

2. **Hacer accesible el nombre en las vistas:**
   - En el archivo principal del servidor (`app.js`), importa el archivo `config.js` y utiliza un middleware para hacerlo accesible en todas las vistas:

```
const config = require('./config');

app.use((req, res, next) => {
  res.locals.title = config.shopName;
  next();
});
```

3. **Actualizar las vistas:**
   - En cada archivo de vista donde se utiliza el nombre de la tienda, reemplázalo por la variable `title (viene así por defecto)`:

```
<%- include("header", {}) %>
<div class="container">
    <h1><%= title %></h1>
    <p>Bienvenido a la tienda <%= title %></p>
```

```
    <p>Aquí podrá encontrar los mejores embutidos y productos gastronómicos
de la región.</p>
    <p>Visite <a href="/tienda">nuestra tienda</a> para ver los productos
que tenemos.</p>
    <img src="/images/embutidos.jpg" alt="Imagen con los embutidos de la
tienda" width="750" height="475">
</div>
<%- include("footer", {}) %
```

4. **Eliminar el nombre hardcodeado en las rutas:**
    - Si tienes rutas que están usando el nombre de la tienda, reemplázalo por la variable global accesible desde `res.locals.shopName`. Por ejemplo en index.js:

```
1. const express = require('express');
2. const router = express.Router();
3.
4. /* GET home page. */
5. router.get('/', function(req, res, next) {
6.   res.render('index', {user:req.session.user, title:res.locals.title});
7. });
8.
9. module.exports = router;
10.
```

Apartado 2

Hay una cuestión muy importante a la que nos obliga la normativa europea y es sobre el uso de cookies en la web. Tenemos que informar al usuario de que estamos usando cookies en nuestra página y que las acepte. Para ello complete las siguientes tareas:

1. Cree una sección que aparezca en la parte inferior de todas las páginas con las siguientes características visuales:

    - Que aparezca un título con un mensaje informando al usuario del uso de cookies.
    - Que tenga un botón de aceptar y otro de rechazar.
    - Que ocupe todo el ancho de la pantalla, tenga el texto y los botones centrados y que esté por encima de cualquier contenido que haya en la página. Tiene que estar siempre visible.
    - Usa CSS puro para crear esta sección, no uses ningún framework.
    - Puedes encontrar una referencia de cómo tendría que quedar en el anexo:

https://programadorwebvalencia.com/Javascript-ejemplo-aviso-o-cartel-de-cookie/

Accedo al footer.ejs:

Creo los botones y html:

```
</body>
<section class="cookies">
    <h2 class="cookies__titulo">¿Aceptas nuestras Cookies?</h2>
    <p class="cookies__texto">Usamos cookies para mejorar tu experiencia en la
web.</p>
    <div class="cookies__botones">
        <button class="cookies__boton cookies__boton--si">Si</button>
        <button class="cookies__boton cookies__boton--no">No</button>

    </div>
</section>
```

Luego aplico estilos caseros:

```
<style>
    .cookies {
  position: fixed;
  bottom: 0;
  left: 0;
  right: 0;
  background: black;
  color: white;
  font-family: arial;
  text-align: center;
}
/*ESTO LO COMENTO PORQUE NO LO PIDE EN SU IMAGEN
.cookies__boton {
  background: initial;
  border: 2px solid white;
  padding: 1rem;
  font-size: 1rem;
```

```
  color: white;
  width: 5rem;
  text-align: center;
} */

button {
    padding: 1rem;
    width: 5rem;
    margin-bottom: 15px;

}

</style>
```

2. Si el usuario le da a cancelar, redirígele a la página de google.es.

Le añado la referencia de Google:

```
<button class="cookies__boton cookies__boton--si"
onclick="location.href='https://www.google.es'">Rechazar</button>
```

3. Si el usuario acepta las cookies, realice lo siguiente:
- Cierre el mensaje.
- Guarde información en la sesión para que, si sigue visitando otras páginas, no se vuelva a mostrar el mensaje.
- Si el usuario está logeado, guarde en su perfil (en la base de datos) información de que ha aceptado las cookies.

Copiamos y pegamos la lógica del JS:

```
<!-- No borrar, aquí se generarán todas las etiquetas <script> si acepta el
usuario -->
    <div id="nuevosScripts"></div>

    <script defer>
    let cookies = () => {

//============================================================================
        // COOKIES

//============================================================================

        //--------------------------------------------------------
        // Configuración
        //--------------------------------------------------------
        const urlsScriptsCookies = ['https://analytics.google.com',
'https://facebook.com'];

        function contenidoScriptsCookies () {
```

```
    ////////////  ¿Google Analítics? /////////////
    ////////////  ¿Facebook Pixel? /////////////
    ////////////  ¿Admob? /////////////
    ////////////  etc  /////////////
}


//-----------------------------------------------------------
// Variables
//-----------------------------------------------------------
const seccionCookie = document.querySelector('section.cookies');
const cookieSi = document.querySelector('.cookies__boton--si');
const cookieNo = document.querySelector('.cookies__boton--no');
const nuevosScripts = document.querySelector('#nuevosScripts');

//-----------------------------------------------------------
// Funciones
//-----------------------------------------------------------

/**
  * Método que oculta la sección de Cookie para siempre
  */
function ocultarCookie() {
    // Borra la sección de cookies en el HTML
    seccionCookie.remove();
}

/**
  * Método que marca las cookies como aceptadas
  */
function aceptarCookies() {
    // Oculta el HTML de cookies
    ocultarCookie();
    // Guarda que ha aceptado
    localStorage.setItem('cookie', true);
    // Tu codigo a ejecutar si aceptan las cookies
    ejecutarSiAcepta();
}

/**
  * Método que marca las cookies como denegadas
  */
function denegarCookies() {
    // Oculta el HTML de cookies
    ocultarCookie();
    // Guarda que ha aceptado
    localStorage.setItem('cookie', false);
    // Redirige a Google
    window.location.href = 'https://www.google.es';
}
```

```
        /**
          * Método que ejecuta tu código si aceptan las cookies
          */
        function ejecutarSiAcepta() {
            // Crea los <script>
            urlsScriptsCookies.forEach((url) => {
                const nuevoScript = document.createElement('script');
                nuevoScript.setAttribute('src', url);
                nuevosScripts.appendChild(nuevoScript);
            });
            // Lanza los códigos
            contenidoScriptsCookies();
        }

        /**
          * Método que inicia la lógica
          */
        function iniciar() {
            // Comprueba si en el pasado el usuario ha marcado una opción
            if (localStorage.getItem('cookie') !== null) {
                if(localStorage.getItem('cookie') === 'true') {
                    // Aceptó
                    aceptarCookies();
                } else {
                    // No aceptó
                    denegarCookies();
                }
            }
        }

        //-----------------------------------------------------------
        // Eventos
        //-----------------------------------------------------------
        cookieSi.addEventListener('click',aceptarCookies, false);
        cookieNo.addEventListener('click',denegarCookies, false);

        return {
            iniciar: iniciar
        }
    }

    // Activa el código. Comenta si quieres desactivarlo.
    cookies().iniciar();

    </script>
```

Posteriormente, añadimos lógica para guardar en la bbdd:

```
async function aceptarCookies() {
            // Oculta el HTML de cookies
            ocultarCookie();
            // Guarda que ha aceptado
```

```
            localStorage.setItem('cookie', true);
             // Lógica para guardar en la base de datos si está logeado
            const isUserLoggedIn = document.body.dataset.userLoggedIn ===
'true';
            if (isUserLoggedIn) {
                try {
                    const response = await fetch('/cookies/accept', { method:
'POST' });
                    if (!response.ok) {
                        console.error('Error al guardar la aceptación de
cookies en la base de datos');
                    }
                } catch (error) {
                    console.error('Error:', error);
                }
            }
            // Tu codigo a ejecutar si aceptan las cookies
            ejecutarSiAcepta();
        }
```

Backend para guardar cookies en la base de datos: Si el usuario está logeado, guarda la decisión de aceptar cookies en la base de datos. Crea una ruta en tu backend:

Añadimos esta funcionalidad:

```
users.data[username] = {username, hash, last_Login: new Date().toISOString,
            cookiesAccepted: false, // Agregamos esta propiedad};

        };
    });
};
```

Y al final del mismo archivo user.models.js:

```
// Método para marcar cookies como aceptadas
users.acceptCookies = function (username) {
    if (!users.data.hasOwnProperty(username)) {
        throw new Error(`El usuario ${username} no existe.`);
    }
    users.data[username].cookiesAccepted = true;
};

// Método para verificar si las cookies han sido aceptadas
users.hasAcceptedCookies = function (username) {
    if (!users.data.hasOwnProperty(username)) {
        return false;
    }
    return users.data[username].cookiesAccepted;
};
```

Ahora, necesitamos una ruta que llame al método acceptCookies del modelo cuando un usuario acepta cookies.

Crear la ruta routes/cookies.js:

```
const express = require('express');
const router = express.Router();
const database = require('../database');

// Ruta para aceptar cookies
router.post('/accept', (req, res) => {
    if (req.session.user && req.session.user.username) {
        try {
            database.user.acceptCookies(req.session.user.username);
            res.status(200).json({ message: 'Cookies aceptadas' });
        } catch (err) {
            console.error(err);
            res.status(500).json({ error: 'No se pudieron aceptar las cookies'
});
        }
    } else {
        res.status(401).json({ error: 'Usuario no autenticado' });
    }
});

module.exports = router;
```

Añadimos la nueva ruta en `app.js`:

```
const cookiesRouter = require('./routes/cookies');

// Otras rutas...

app.use('/cookies', cookiesRouter);
```

4. Cuando un usuario hace login en nuestra página, si en la sesión actual ya ha aceptado cookies o en su perfil hemos guardado previamente información de que ha aceptado las cookies, no debería mostrarse el mensaje. En caso contrario sí que debería aparecer.

Para ver resultado casi perfecto, acceder al repo…

Sockets.io

https://www.luisllamas.es/como-usar-socketio-con-nodejs/?utm_
https://programandoweb.net/sencillo-chat-con-node-js-express-socket-htaccess/?utm

Para **centralizar la configuración** y utilizar variables de entorno, vamos a seguir estos pasos. Esto mejora la seguridad y el mantenimiento de tu proyecto.

---

1. Instala `dotenv` para manejar variables de entorno

En la terminal, ejecuta el siguiente comando para instalar **dotenv**:

```
npm install dotenv
```

---

2. Crea un archivo `.env` en la raíz del proyecto

Aquí es donde almacenarás los valores críticos como el puerto, nombre de la tienda y credenciales de la base de datos.

Contenido de `.env`:

```
PORT=4000
SHOP_NAME=Embutidos León
DB_USER=admin
DB_PASSWORD=supersecurepassword
DB_HOST=localhost
DB_NAME=carrito_db
SECRET_KEY=UnaFraseMuySecreta
```

---

3. Crea un archivo `config.js` para cargar la configuración

Este archivo centraliza todas las configuraciones y las lee desde el archivo `.env`.

`config.js`:

```
require('dotenv').config();

module.exports = {
    port: process.env.PORT || 3000,
    shopName: process.env.SHOP_NAME || 'Tienda Default',
    db: {
        user: process.env.DB_USER || 'root',
        password: process.env.DB_PASSWORD || '',
        host: process.env.DB_HOST || 'localhost',
        name: process.env.DB_NAME || 'test_db'
    },
    sessionSecret: process.env.SECRET_KEY || 'defaultSecret'
};
```

---

4. Actualiza `app.js` para usar la configuración centralizada

Importa el archivo `config.js` y reemplaza los valores "hardcodeados" por valores centralizados.

Modificaciones en `app.js`:

```
const createError = require('http-errors');
const express = require('express');
const path = require('path');
const cookieParser = require('cookie-parser');
const logger = require('morgan');
const session = require('express-session');
const config = require('./config'); // Importa la configuración
centralizada

const indexRouter = require('./routes/index');
const loginRouter = require('./routes/login');
const carritoRouter = require('./routes/carrito');

const app = express();

// Configuración del puerto desde variables de entorno
const PORT = config.port;

// Configuración del servidor HTTP
const http = require('http');
const server = http.createServer(app);

// Configuración de sesión
app.use(session({
    secret: config.sessionSecret, // Secreto centralizado
    resave: false,
    saveUninitialized: true
}));

// Middleware para pasar el nombre de la tienda a las vistas
app.use((req, res, next) => {
    res.locals.title = config.shopName; // Nombre centralizado
    next();
});

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/login', loginRouter);
app.use('/carrito', carritoRouter);

// Catch 404 and forward to error handler
```

```
app.use(function(req, res, next) {
    next(createError(404));
});

// Error handler
app.use(function(err, req, res, next) {
    res.locals.message = err.message;
    res.locals.error = req.app.get('env') === 'development' ? err :
{};

    res.status(err.status || 500);
    res.render('error');
});

// Iniciar el servidor
server.listen(PORT, () => {
    console.log(`Servidor iniciado en http://localhost:${PORT}`);
});
```

---

5. Acceder a la configuración en otros archivos

Si necesitas acceder a valores como la configuración de la base de datos, puedes importar `config.js` en otros archivos.

Ejemplo en `database/index.js`:

```
const config = require('../config');

console.log('Conectando a la base de datos...');
console.log(`Usuario: ${config.db.user}`);
console.log(`Host: ${config.db.host}`);
```

Beneficios de esta configuración

1. **Centralización**: Toda la configuración está en un solo lugar (`.env` y `config.js`).
2. **Seguridad**: Las credenciales sensibles están en `.env` y no en el código fuente.
3. **Facilidad de mantenimiento**: Si necesitas cambiar un valor, solo editas el archivo `.env`.

Prueba final

1. Crea un archivo `.env` con los valores críticos.
2. Asegúrate de que `config.js` carga correctamente las variables.
3. Ejecuta el servidor:

```
npm start
```

Deberías ver en la consola:
`Servidor iniciado en http://localhost:4000`

1. Estructura general de Express

Tu proyecto está organizado de manera clara y funcional. La estructura típica de un proyecto Express tiene:

```
/bin           --> Punto de entrada del servidor
/database      --> Lógica de base de datos (modelos y funciones)
  ├─ models    --> Definición de modelos (usuarios, carrito)
/public        --> Archivos estáticos (CSS, imágenes, JS)
  ├─ images    --> Imágenes del proyecto
  ├─ stylesheets --> Archivos CSS
/routes        --> Definición de rutas de la aplicación
/views         --> Plantillas EJS para renderizar vistas dinámicas
app.js         --> Configuración principal de la aplicación Express
config.js      --> Configuración centralizada (puertos, claves,
variables de entorno)
```

---

2. Detalle de cada carpeta y archivo

1. `/bin/www`

- **Función**: Es el **punto de entrada** del servidor Node.js.
- Aquí se crea la instancia de HTTP y se importa `app.js` para iniciar el servidor.
- **Ejemplo**:

```
const app = require('../app'); // Importa app.js
const http = require('http');

const port = process.env.PORT || '4000';
const server = http.createServer(app);

server.listen(port, () => {
    console.log(`Servidor corriendo en
http://localhost:${port}`);
});
```

---

2. `/routes`

- **Función**: Define las rutas del servidor. Cada archivo maneja una funcionalidad específica.

- **Archivos comunes**:
  - **`carrito.js`**: Rutas para manejar el carrito de la compra (añadir, eliminar, vaciar).
  - **`login.js`**: Rutas de autenticación (login, logout).
  - **`tienda.js`**: Rutas relacionadas con la tienda (productos y comentarios).
  - **`index.js`**: Ruta principal (página de inicio).
  - **`cookies.js`**: Manejo de políticas de cookies.
  - **`restricted.js`**: Acceso a rutas restringidas.
  - **Ejemplo de ruta en `carrito.js`**:

```
router.post('/add', (req, res) => {
    const { productId } = req.body;
    database.cart.addToCart(req.session.user.username,
productId);
    res.redirect('/carrito');
});
```

---

3. `/views`

- **Función**: Contiene las plantillas EJS que renderizan el HTML dinámico.
- **Archivos**:
  - **`header.ejs`**: Cabecera de la página (barra de navegación).
  - **`footer.ejs`**: Pie de página común.
  - **`tienda.ejs`**: Vista de la tienda donde se listan los productos.
  - **`carrito.ejs`**: Vista del carrito de compras.
  - **`error.ejs`**: Página de error (404, 500, etc.).
  - **`login.ejs`**: Vista de inicio de sesión.
- **Ejemplo de plantilla EJS (`tienda.ejs`)**:

```
<%- include("header", {}) %>
<h1>Tienda</h1>
<% productos.forEach(product => { %>
    <div>
        <h2><%= product.nombre %></h2>
        <img src="<%= product.imagen %>" alt="Imagen de <%=
product.nombre %>">
    </div>
<% }) %>
<%- include("footer", {}) %>
```

---

4. `/public`

- **Función**: Archivos estáticos que el cliente puede usar directamente.
  - **`images/`**: Contiene imágenes.
  - **`stylesheets/style.css`**: Archivos CSS para dar estilo al proyecto.

---

5. `/database`

- **Función**: Maneja la lógica relacionada con la base de datos y los modelos.

- **Archivos**:
  - **`index.js`**: Inicializa la base de datos con datos de prueba (como usuarios y carritos).
  - **`models/`**:
    - **`user.model.js`**: Lógica de usuario (registro, login, cookies).
    - **`carrito.model.js`**: Lógica del carrito (añadir, eliminar productos).
- **Ejemplo de `user.model.js`**:

```
users.register = function(username, password) {
    users.data[username] = { username, passwordHash:
hashPassword(password) };
};
```

---

6. `app.js`

- **Función**: Archivo principal de configuración de Express.
- Configura **middlewares**, rutas, sesiones, y el servidor **Socket.IO**.
- **Componentes importantes**:
  - **Configuración de vistas**:

```
javascript
Copiar código
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');
```

  - **Uso de middlewares**:

```
javascript
Copiar código
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(express.static(path.join(__dirname, 'public')));
```

  - **Manejo de sesiones**:

```
javascript
Copiar código
app.use(session({
    secret: "Una frase muy secreta",
    resave: false,
    saveUninitialized: true
}));
```

  - **Rutas**:

```
javascript
Copiar código
app.use('/carrito', carritoRouter);
app.use('/tienda', tiendaRouter);
```

---

7. `config.js`

- **Función**: Centraliza configuraciones importantes como el puerto, nombre de la tienda, o claves secretas.
- **Ejemplo**:

```
javascript
Copiar código
require('dotenv').config();

module.exports = {
    port: process.env.PORT || 4000,
    shopName: 'Embutidos León',
    sessionSecret: 'UnaFraseMuySecreta'
};
```

Flujo de la Aplicación

1. **Arranque**:
   - El servidor inicia en `/bin/www`.
   - Importa `app.js` que configura la aplicación Express y las rutas.
2. **Rutas**:
   - Las rutas (`/routes`) manejan las solicitudes HTTP del cliente.
   - Cada ruta llama a la lógica correspondiente en `/database`.
3. **Vistas**:
   - Se renderizan dinámicamente usando **EJS** desde la carpeta `/views`.
4. **Datos**:
   - Se gestionan y almacenan en **`/database`**, que usa modelos como `user.model.js` y `carrito.model.js`.
5. **Sockets**:
   - **`app.js`** configura **Socket.IO** para manejar eventos en tiempo real (comentarios y carrito).

Resumen

- **`/routes`**: Maneja las rutas y la lógica de cada funcionalidad.
- **`/views`**: Renderiza las plantillas HTML dinámicas con EJS.
- **`/database`**: Modelos y lógica de datos.
- **`/public`**: Recursos estáticos (CSS, imágenes).
- **`app.js`**: Configura Express, sesiones, rutas y sockets.
- **`config.js`**: Configuración global del proyecto.