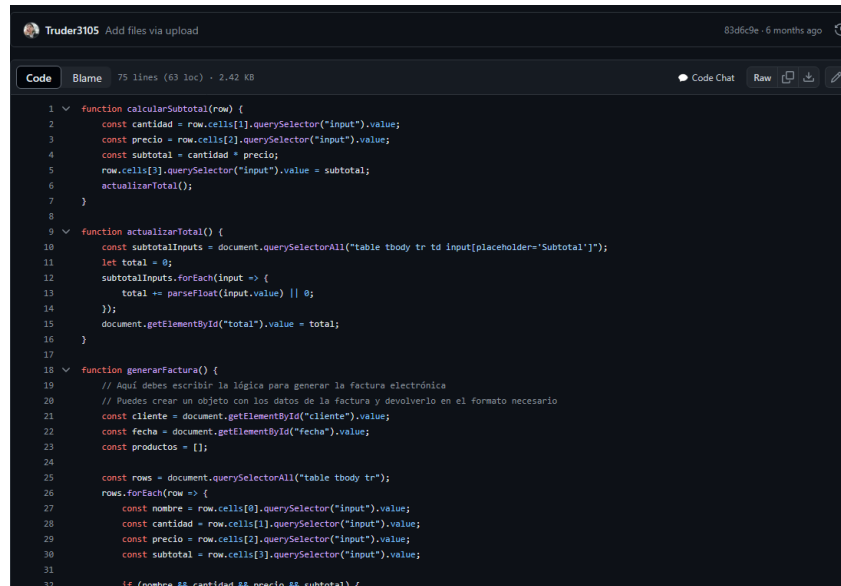


## Arquitectura de diseño y patrones de diseño:

JS:



```
1 function calcularSubtotal(row) {
2   const cantidad = row.cells[1].querySelector("input").value;
3   const precio = row.cells[2].querySelector("input").value;
4   const subtotal = cantidad * precio;
5   row.cells[3].querySelector("input").value = subtotal;
6   actualizarTotal();
7 }
8
9 function actualizarTotal() {
10  const subtotalInputs = document.querySelectorAll("table tbody tr td input[placeholder='Subtotal']");
11  let total = 0;
12  subtotalInputs.forEach(input => {
13    total += parseFloat(input.value) || 0;
14  });
15  document.getElementById("total").value = total;
16 }
17
18 function generarFactura() {
19  // Aquí debes escribir la lógica para generar la factura electrónica
20  // Puedes crear un objeto con los datos de la factura y devolverlo en el formato necesario
21  const cliente = document.getElementById("cliente").value;
22  const fecha = document.getElementById("fecha").value;
23  const productos = [];
24
25  const rows = document.querySelectorAll("table tbody tr");
26  rows.forEach(row => {
27    const nombre = row.cells[0].querySelector("input").value;
28    const cantidad = row.cells[1].querySelector("input").value;
29    const precio = row.cells[2].querySelector("input").value;
30    const subtotal = row.cells[3].querySelector("input").value;
31
32    if (nombre && cantidad && precio && subtotal) {
```

**1. Patrón Módulo:** Las funciones `calcularSubtotal`, `actualizarTotal`, `generarFactura` y `enviarFactura` pueden considerarse como parte de un patrón módulo, donde cada función encapsula una funcionalidad específica.

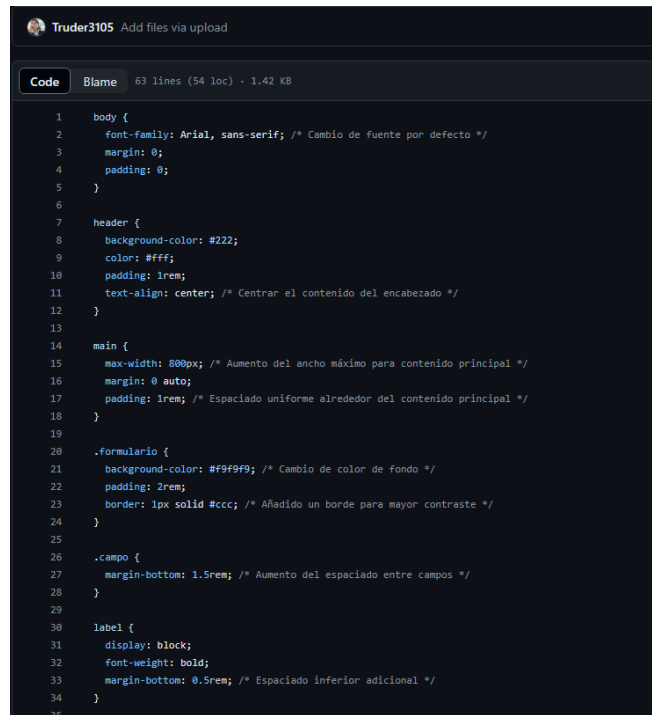
**2. Patrón de Factoría:** Aunque no se muestra explícitamente, la función `generarFactura` podría estar utilizando un patrón de factoría implícito al crear y retornar un objeto factura basado en los datos recogidos de la interfaz de usuario.

**3. Patrón Singleton:** El manejo del total en `actualizarTotal` sugiere un patrón singleton, ya que parece haber una única instancia del total que se actualiza a lo largo de la aplicación.

**4. Patrón de Comando:** La función `enviarFactura` utiliza el patrón de comando al encapsular la solicitud de envío de la factura como una acción que se puede ejecutar.

En cuanto a la **arquitectura de diseño**, el código refleja una arquitectura de cliente-servidor típica de las aplicaciones web, donde el navegador (cliente) interactúa con el servidor a través de solicitudes HTTP. Además, se puede inferir una arquitectura de **capas**, con una separación entre la lógica de presentación (interfaz de usuario), la lógica de negocio (cálculo de subtotales y totales, generación de factura) y la capa de persistencia/datos (envío de la factura al servidor).

## CSS

A screenshot of a code editor interface. At the top, there's a header bar with a user profile icon and the name 'Truder3105', followed by a button 'Add files via upload'. Below this is a tab bar with 'Code' selected and 'Blame' next to it. To the right of the tabs, it says '63 lines (54 loc) · 1.42 KB'. The main area shows CSS code with line numbers from 1 to 34. The code defines styles for 'body', 'header', 'main', '.formulario', '.campo', and 'label'. Comments in Spanish explain the purpose of various styles like font-family, padding, margin, and background-color. The code uses rem units for spacing and auto for margins to create a responsive layout.

```
1 body {
2   font-family: Arial, sans-serif; /* Cambio de fuente por defecto */
3   margin: 0;
4   padding: 0;
5 }
6
7 header {
8   background-color: #222;
9   color: #fff;
10  padding: 1rem;
11  text-align: center; /* Centrar el contenido del encabezado */
12 }
13
14 main {
15   max-width: 800px; /* Aumento del ancho máximo para contenido principal */
16   margin: 0 auto;
17   padding: 1rem; /* Espaciado uniforme alrededor del contenido principal */
18 }
19
20 .formulario {
21   background-color: #f9f9f9; /* Cambio de color de fondo */
22   padding: 2rem;
23   border: 1px solid #ccc; /* Añadido un borde para mayor contraste */
24 }
25
26 .campo {
27   margin-bottom: 1.5rem; /* Aumento del espaciado entre campos */
28 }
29
30 label {
31   display: block;
32   font-weight: bold;
33   margin-bottom: 0.5rem; /* Espaciado inferior adicional */
34 }
```

- **Separación de preocupaciones:** Cada elemento de la página (cuerpo, encabezado, contenido principal, formulario, campos, etc.) tiene su propio conjunto de estilos, lo que facilita el mantenimiento y la escalabilidad.

- **Diseño Responsivo:** El uso de `max-width` y los márgenes automáticos en el elemento principal sugiere un enfoque de diseño responsivo, que permite que la página se adapte a diferentes tamaños de pantalla.

- **Patrón de Diseño de Formulario:** Los estilos aplicados a los elementos del formulario (`formulario`, `campo`, `label`, `input`, `button`) indican un patrón de diseño de formulario que busca mejorar la experiencia del usuario al hacer que los formularios sean fáciles de leer y utilizar.

- **Interactividad del Usuario:** La transición en el botón y el cambio de color al pasar el cursor (`hover`) mejoran la interactividad, proporcionando retroalimentación visual al usuario.

En cuanto a la **arquitectura de diseño**, el código refleja una estructura de **diseño por capas** típica de las aplicaciones web, donde la presentación, la lógica y los datos están

separados. En este caso, el CSS se ocupa de la capa de presentación, definiendo cómo se deben mostrar los elementos en la pantalla.

## HTML

```
Truder3105 Add files via upload

Code Blame 106 lines (100 loc) · 3.59 KB

1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <title>Facturación electrónica</title>
6   <link rel="stylesheet" href="style.css">
7 </head>
8 <body>
9   <header>
10    <h1>Facturación electrónica</h1>
11  </header>
12  <main>
13    <section class="formulario">
14      <form id="FacturaForm" onsubmit="event.preventDefault(); enviarFactura();">
15        <div class="campo">
16          <label for="cliente">Cliente</label>
17          <input type="text" id="cliente" placeholder="NIT del cliente" required>
18        </div>
19        <div class="campo">
20          <label for="fecha">Fecha</label>
21          <input type="date" id="fecha" required>
22        </div>
23        <div class="campo">
24          <label for="productos">Productos</label>
25          <table>
26            <thead>
27              <tr>
28                <th>Nombre</th>
29                <th>Cantidad</th>
30                <th>Precio</th>
31                <th>Subtotal</th>
32              </tr>
33            </thead>
34            <tbody>
35              <tr>
36                <td><input type="text" placeholder="Nombre del producto" required></td>
37                <td><input type="number" placeholder="Cantidad" required></td>
38                <td><input type="number" placeholder="Precio" required></td>
39                <td><input type="number" placeholder="Subtotal" readonly></td>
```

## Arquitectura de Diseño:

- **Modelo-Vista-Controlador (MVC):** Aunque no está explícitamente definido, el código sugiere una separación de preocupaciones similar a MVC:

- **Modelo:** La lógica de negocio y los datos de la factura podrían considerarse el modelo.
- **Vista:** El HTML y el CSS corresponden a la vista, definiendo la presentación y la interfaz de usuario.
- **Controlador:** Las funciones de JavaScript actúan como controladores, manejando la lógica de interacción del usuario y la actualización de la vista.

## Patrones de Diseño:

- **Patrón de Módulo:** Las funciones ``calcularSubtotal``, ``actualizarTotal``, y ``enviarFactura`` encapsulan comportamientos específicos, siguiendo el patrón de módulo.
- **Patrón de Comando:** La función ``enviarFactura`` encapsula la acción de enviar la factura, que es un ejemplo del patrón de comando.
- **Patrón de Observador:** Aunque no se muestra directamente, el patrón de observador podría estar presente si las funciones responden a eventos del DOM, como cambios en los inputs o clicks en los botones.

El código refleja prácticas de **programación orientada a eventos**, donde las acciones del usuario (como clics o envíos de formularios) desencadenan funciones específicas. Además, la estructura tiene un enfoque de **desarrollo front-end**, con interacciones del lado del cliente y comunicación con el servidor a través de AJAX para el envío de datos.

(PARA PODER VER LOS CODIGO COMPLETOS ESTOS SE ENCUENTRAN EN EL REPOSITORIO DE GITHUB "WEB-CODE", QUE SE ENCUENTRA TANTO EN EL DOCUMENTO IEEE, COMO EN LOS ANEXOS)