MANUAL DE PROGRAMADOR

PROYECTO FINAL DE SISTEMAS

DOCENTE A CARGO:

JOSE GERARDO GOMEZ VELAZQUEZ

INTEGRANTES DEL EQUIPO:

RUBEN DAVID GARCIA HERNANDEZ

JEYSON MIGUEL CACERES ULLOA

KEVIN BRENES RAMIREZ

PROYECTO:

SISTEMA CENTRALIZADO DE GESTION ESTUDIANTIL A MEDIDA

INDICE

INTRODUCCIÒN 3
Instalación 4
1 instalación de virtualbox 4
2 descargar Debian 6
3 instalación y configuración de Debian 8
Estructura de la BD 19
Muestra de la declaración de nuestras tablas 20
Codigo 23
<!-- Login --> 23
<!----- Panel de estudiantes -----> 24
Panel de profesor 29
<!----- Panel de linuxero maestro -----> 32
Logica JS del archivo index.html 33
Estilos tabla dinámica 34
function getBackendIP 36
cambia paginas 36
CODIGO ESTUDIANTES 41
CODIGO PROFESORES 46

INTRODUCCIÒN

Este proyecto tiene como objetivo el desarrollo de una aplicación web para el control escolar, diseñada para facilitar la gestión de información académica, como los datos de estudiantes, materias, calificaciones y asistencia. La solución ha sido implementada con un enfoque modular, donde la interfaz de usuario interactúa con un servidor backend que se comunica con una base de datos para almacenar y recuperar los datos relevantes.

La arquitectura de la aplicación se basa en un entorno virtualizado, utilizando VirtualBox como herramienta para montar un servidor con Debian como sistema operativo. Este servidor se configura con la dirección IP local para facilitar su acceso desde otros dispositivos en la red, permitiendo a los usuarios interactuar con el sistema desde diferentes ubicaciones dentro de la misma red.

La aplicación fue desarrollada utilizando Node.js como entorno de ejecución de JavaScript, permitiendo la creación de un backend rápido y escalable. En cuanto al almacenamiento de datos, se empleó MariaDB, un sistema de gestión de bases de

datos basado en MySQL, para gestionar la información de manera eficiente y segura.

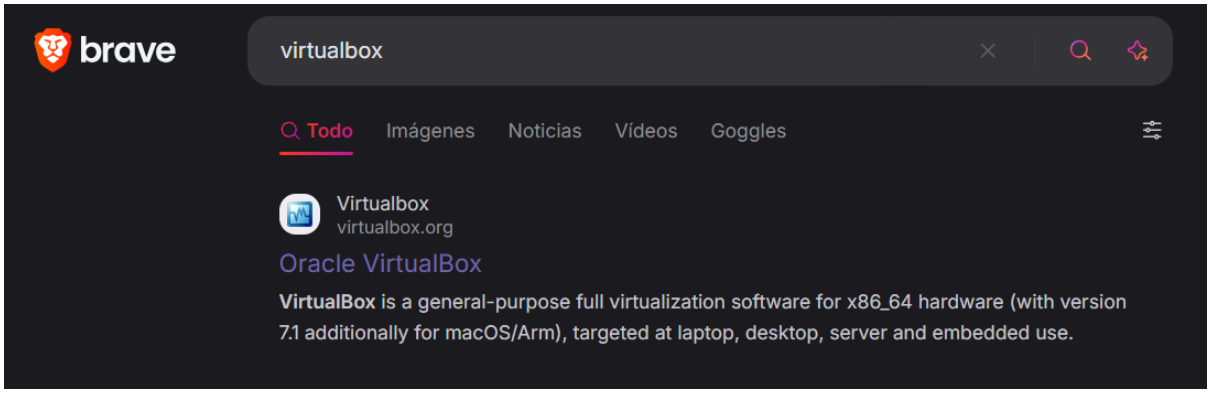
Este manual está diseñado para guiar a los programadores que deseen entender la estructura y funcionamiento del sistema, así como para aquellos interesados en realizar modificaciones o contribuciones adicionales al proyecto. Aquí se detallarán los pasos necesarios para la instalación, configuración del entorno y cómo utilizar la aplicación.

Instalación

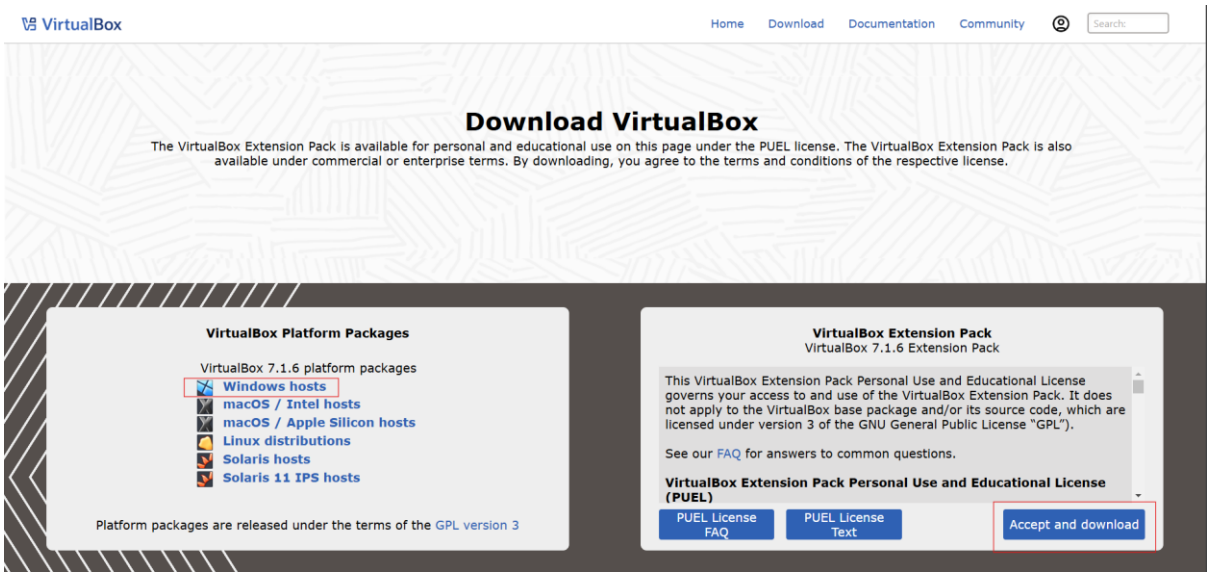
Como ya se mencionó anteriormente este proyecto uso una máquina virtual para montar un servidor para poder trabajar el backend y la redirección al servidor por medio de una red local, si se cuenta con un equipo al que se le puede instalar un sistema operativo diferente al que trae por defecto, se puede prescindir de la máquina virtual e ir a los siguientes pasos

1 instalación de virtualbox

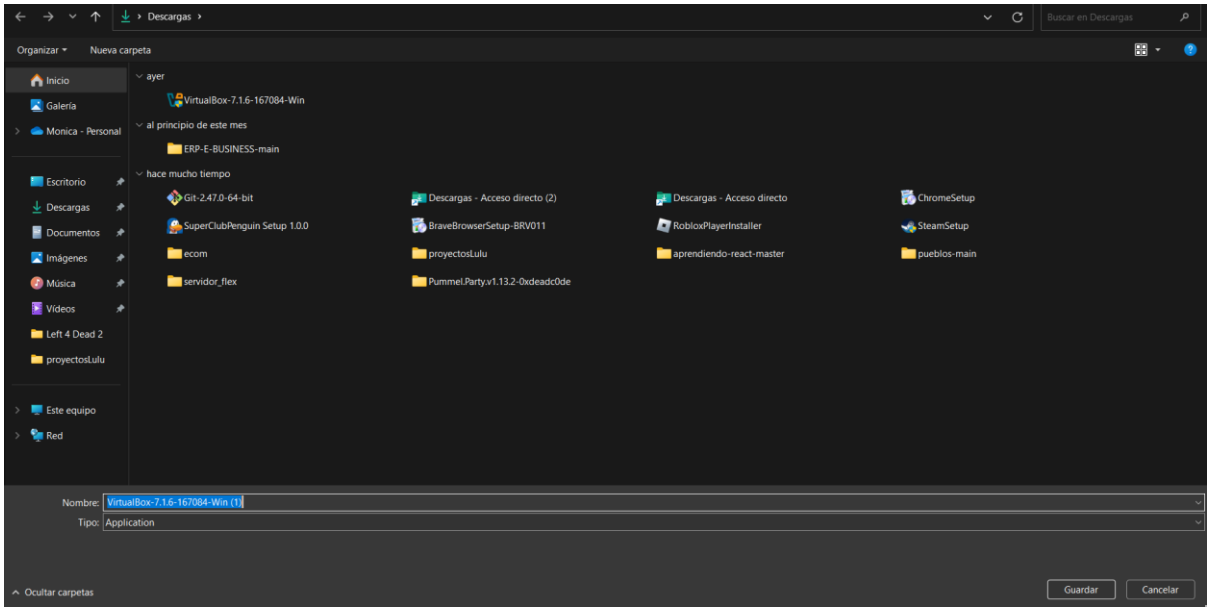
1.1 - en nuestro navegador de preferencia buscaremos “virtualbox” entraremos al primer enlace



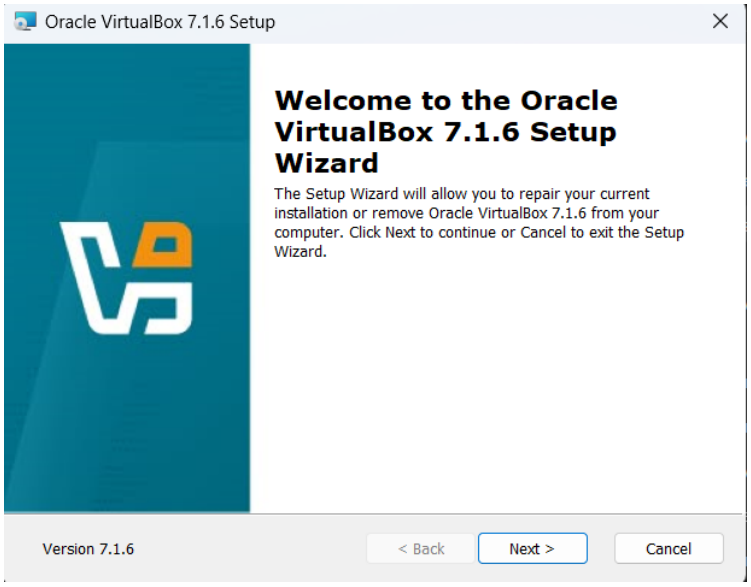
1.2.- una vez dentro de la página descargaremos el paquete de plataforma Windows y el paquete de extensión



Seleccionamos el lugar de almacenamiento

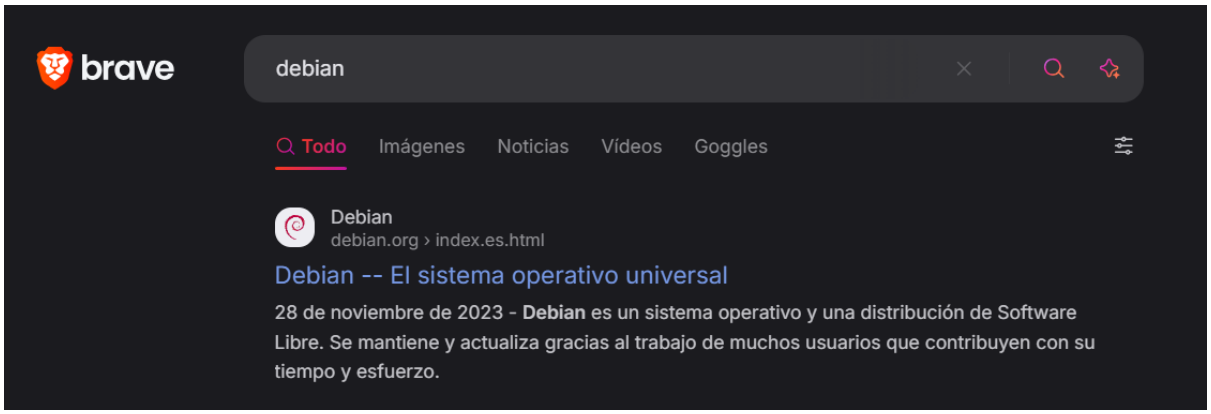


1.3 .- instalamos virtual box



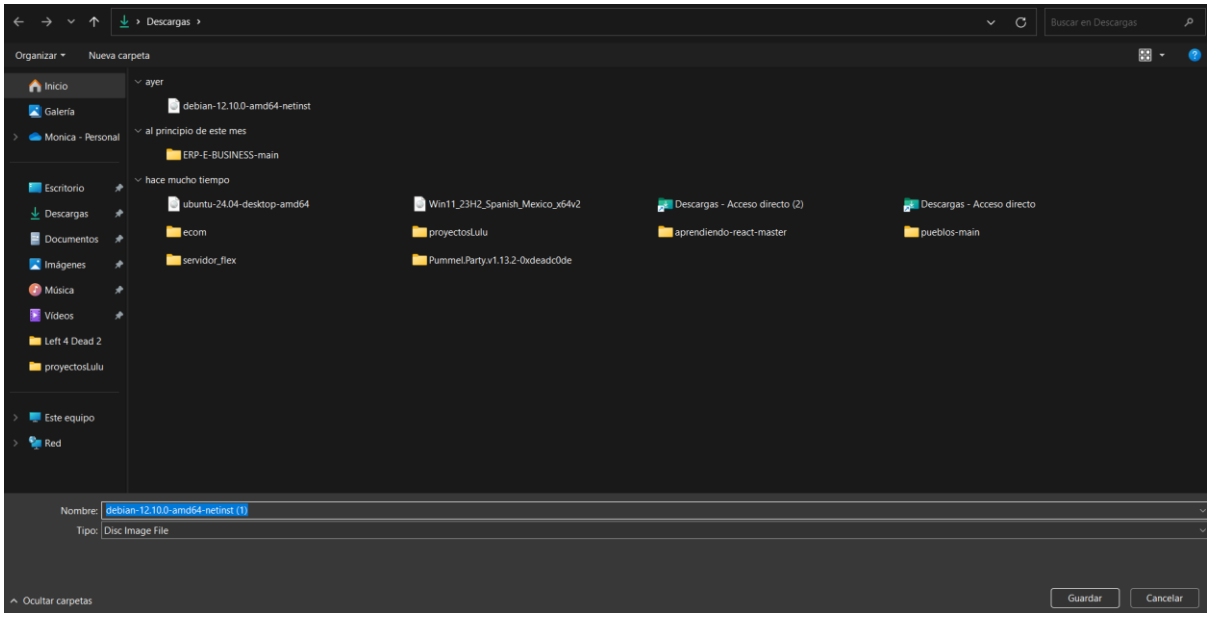
2 descargar Debian

En nuestro navegador de preferencia buscamos “Debian” e ingresamos al primer enlace



Dentro de la página le damos clic al botón de descarga

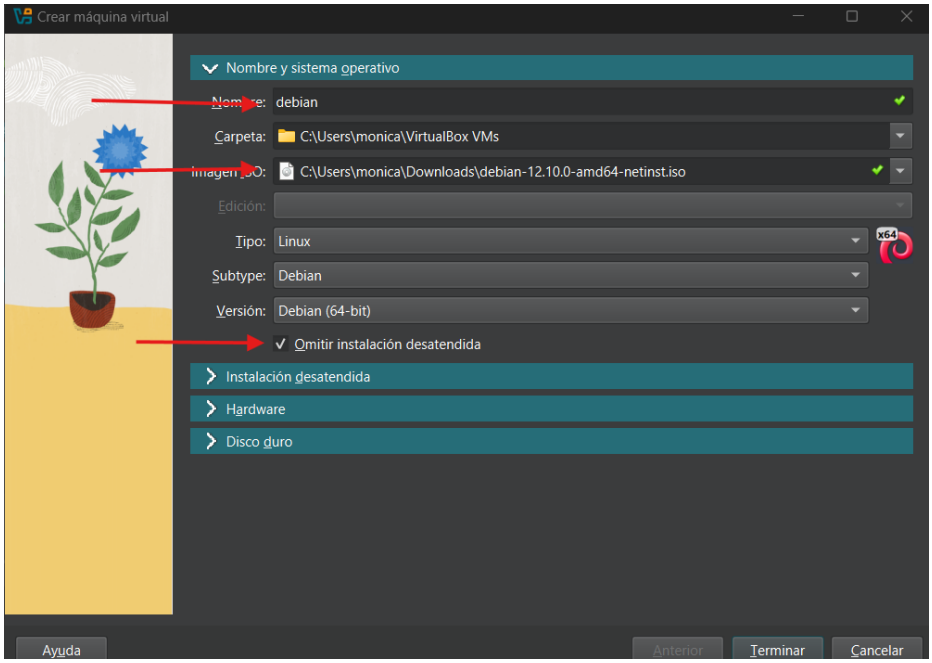
Y seleccionamos la ruta de descarga



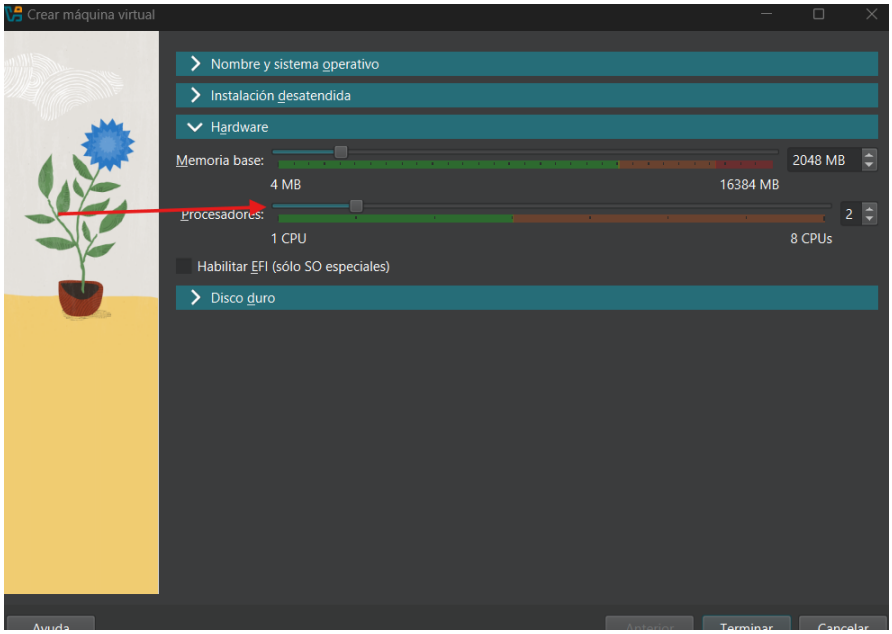
3 instalación y configuración de Debian

Una vez terminados los pasos anteriores ejecutamos VirtualBox y seleccionamos la siguiente opción

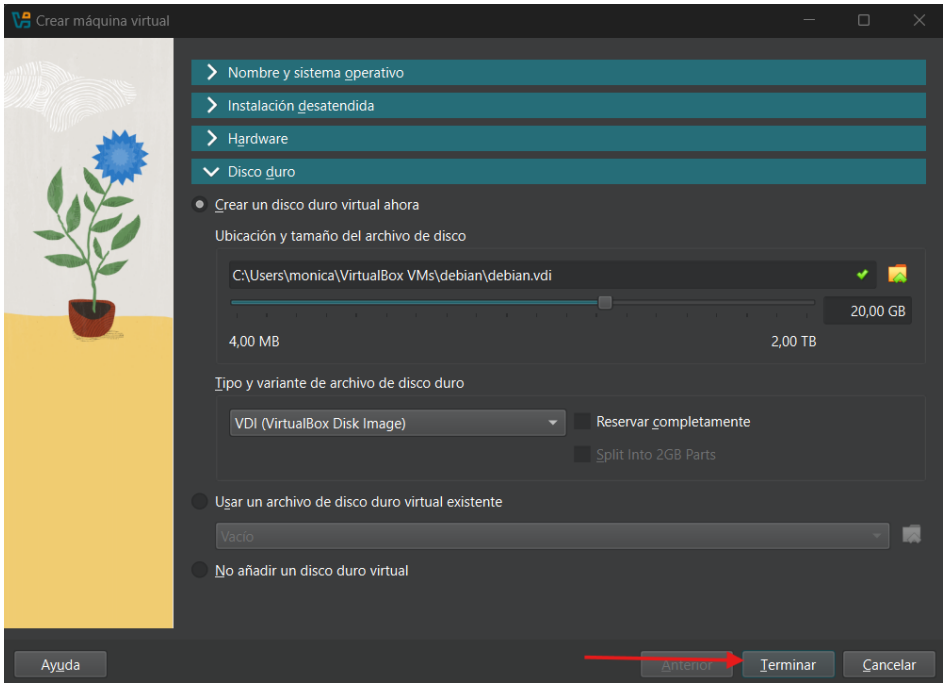
Una vez ahí ingresamos el nombre de nuestra máquina virtual, seleccionamos la iso que descargamos y marcamos la opción de “omitir instalación desatendida” lo demas se queda por defecto



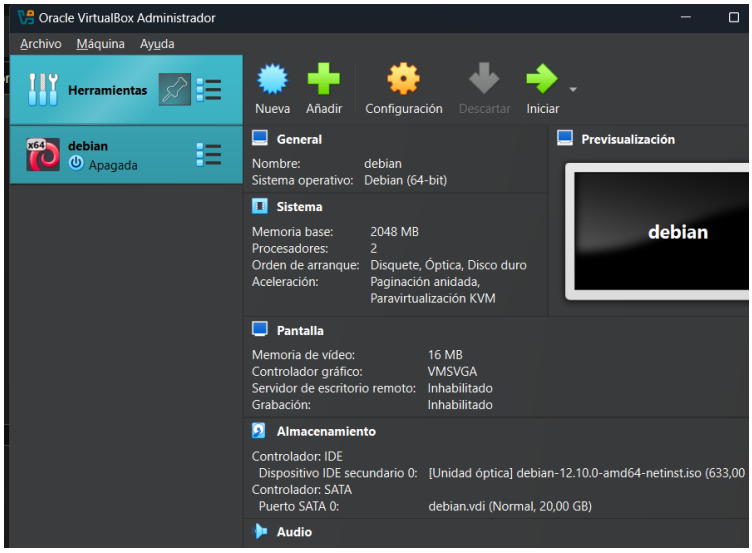
Aumentamos los procesadores a 2



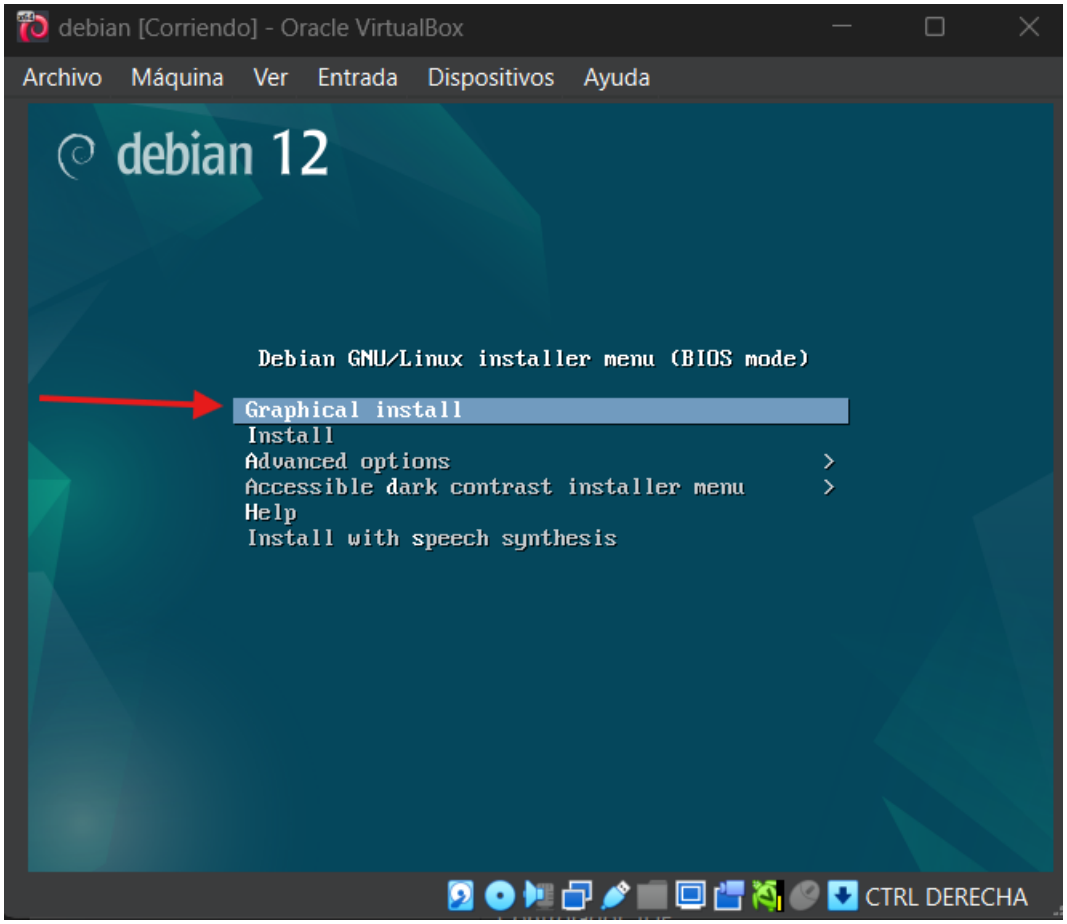
La siguiente opción se queda igual y pulsamos en terminar



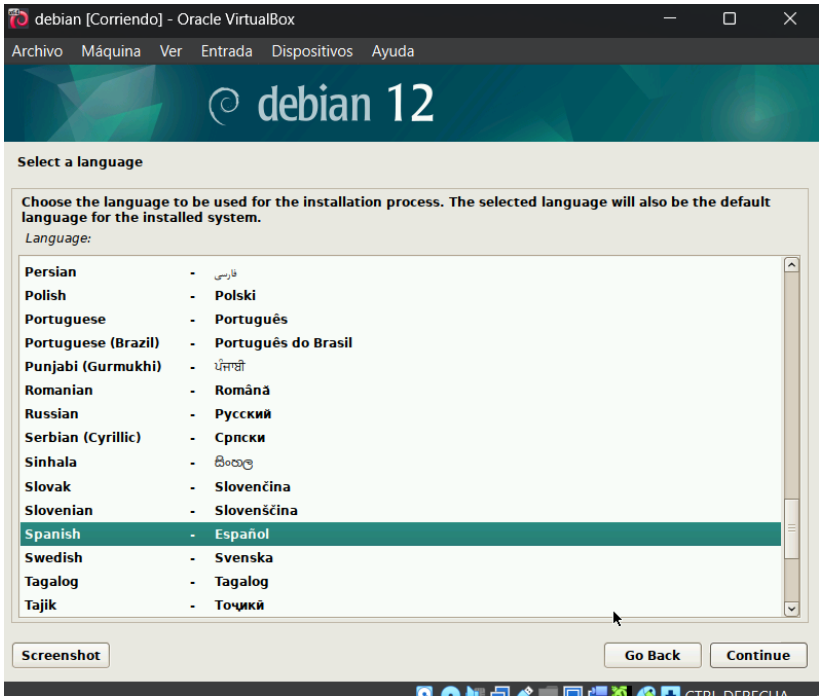
Se mostrara la iso cargada en virtualbox



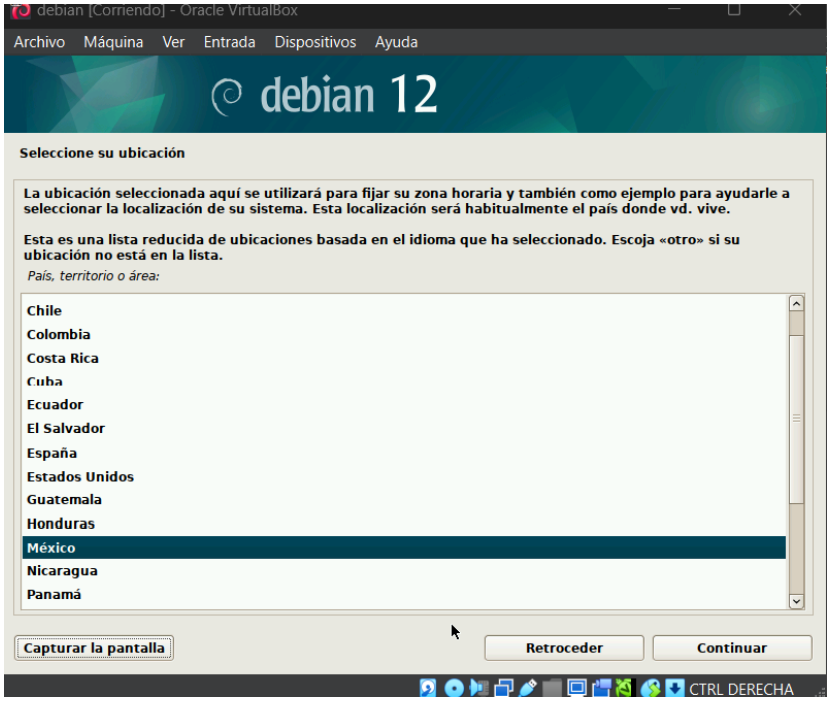
Iniciamos nuestra iso y estando dentro seleccionamos “graphical install”



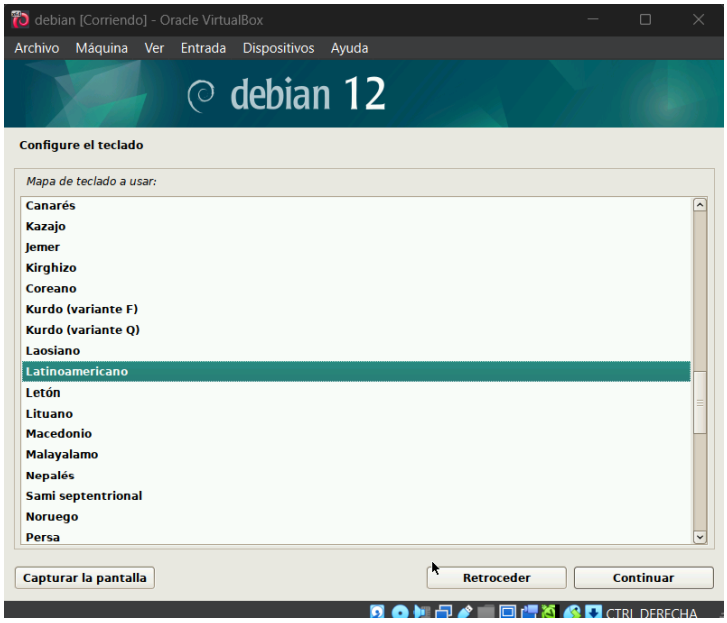
Seleccionamos el idioma de nuestra preferencia



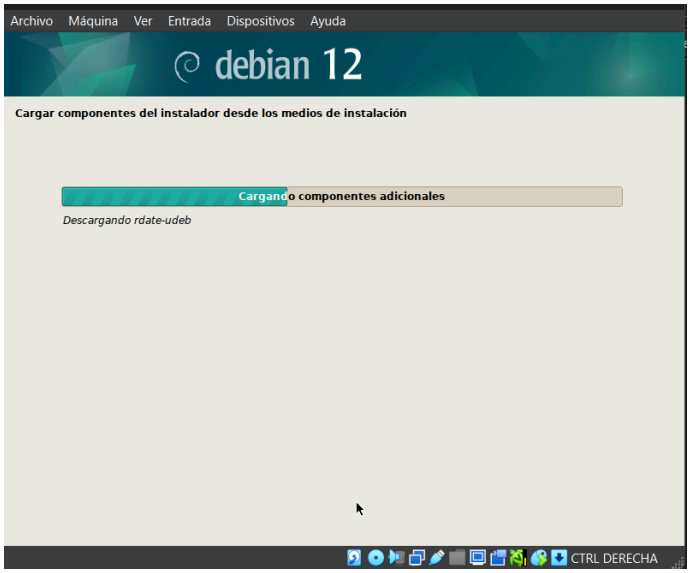
Seleccionamos nuestra ubicación



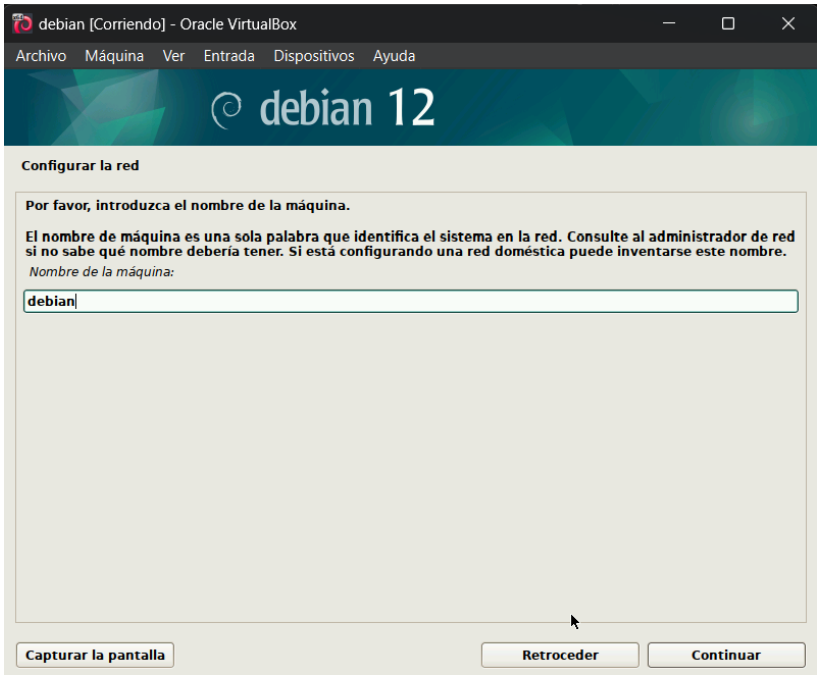
Seleccionamos el tipo de teclado que tenemos



Se empezarán a cargar los componentes

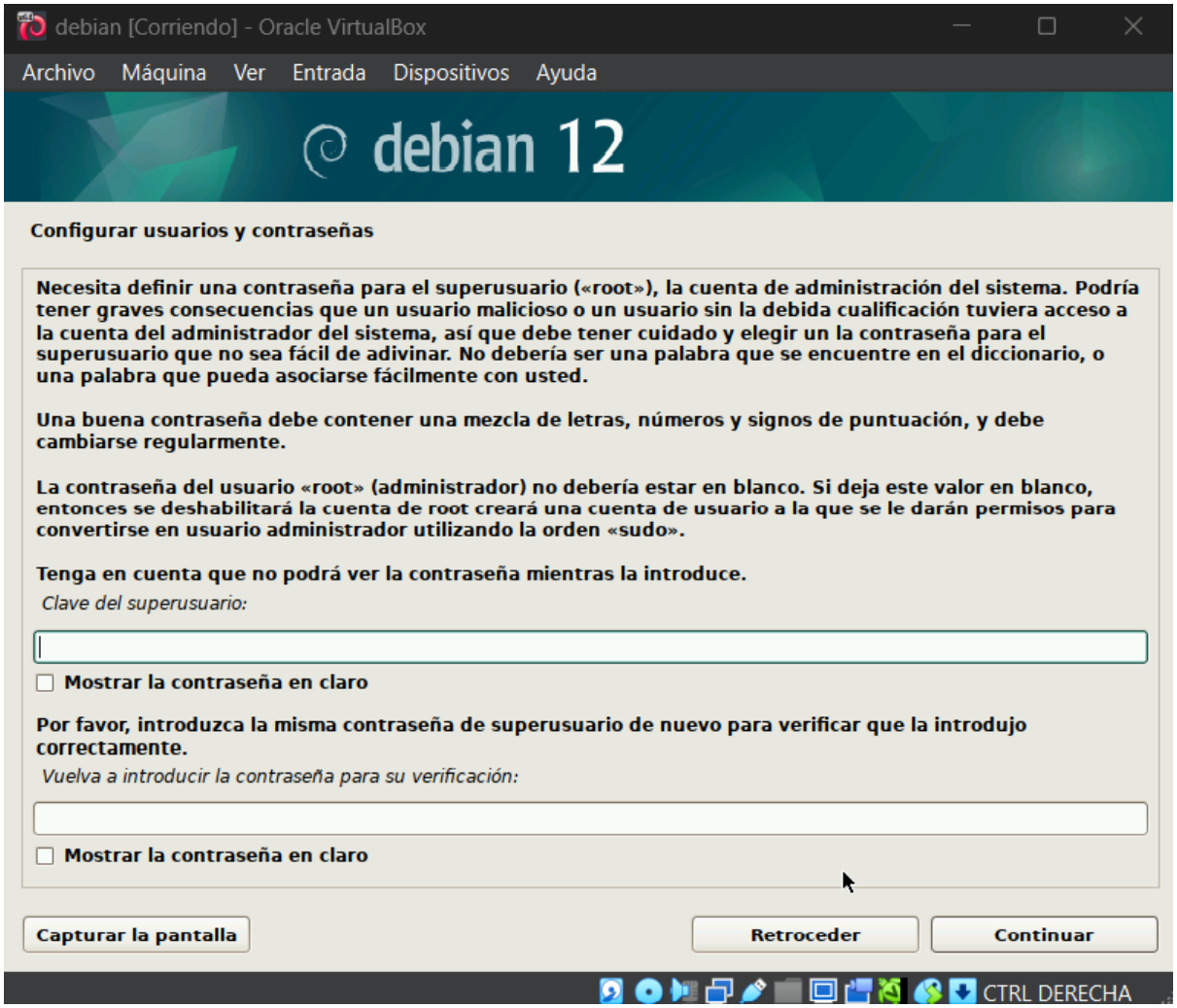


Determinamos el nombre de la red, en este caso lo dejamos como “debian”



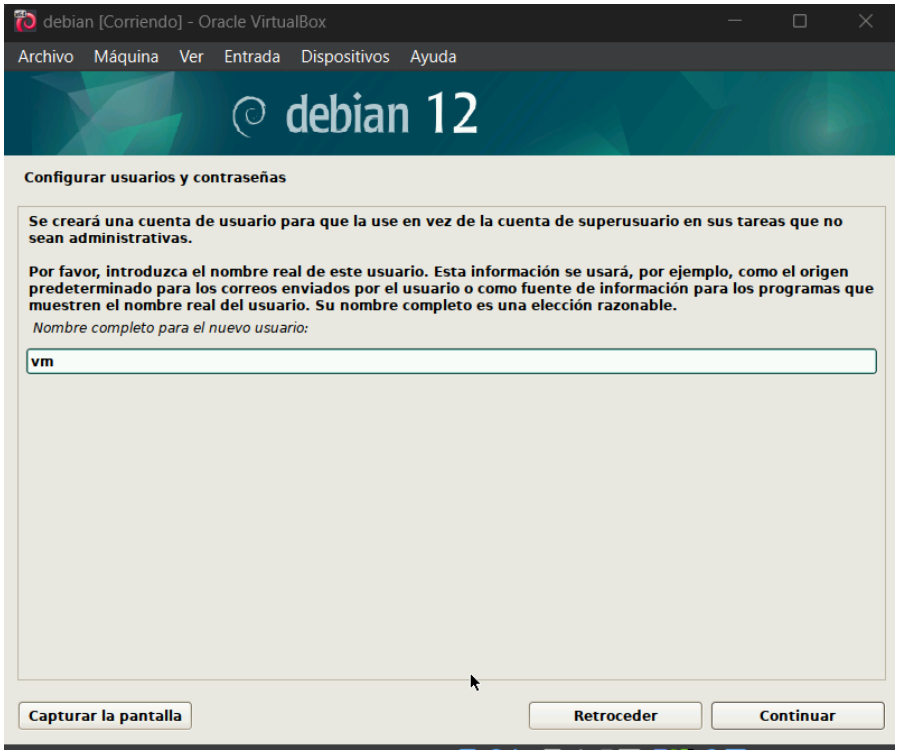
La siguiente opcion se queda en blanco

En la contraseña ingresaremos una que nos acordemos, a gusto personal

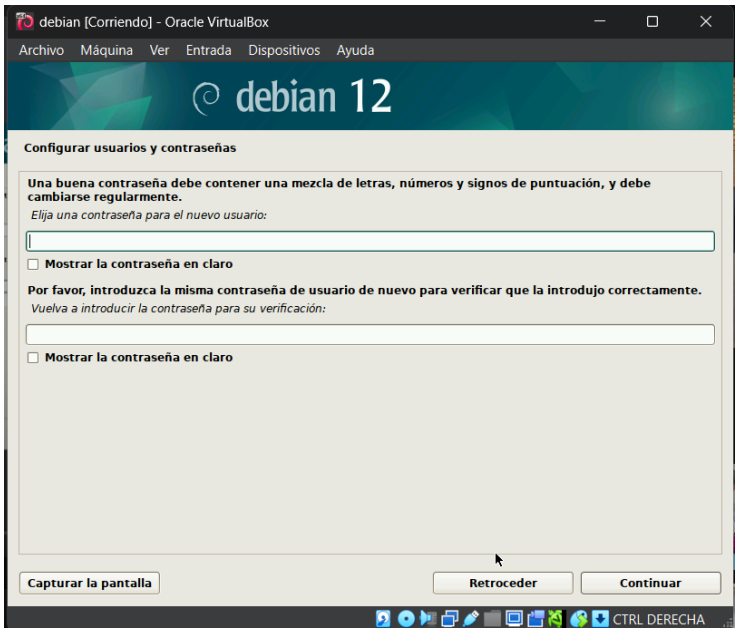


De aqui en adelante imitemos los pasos

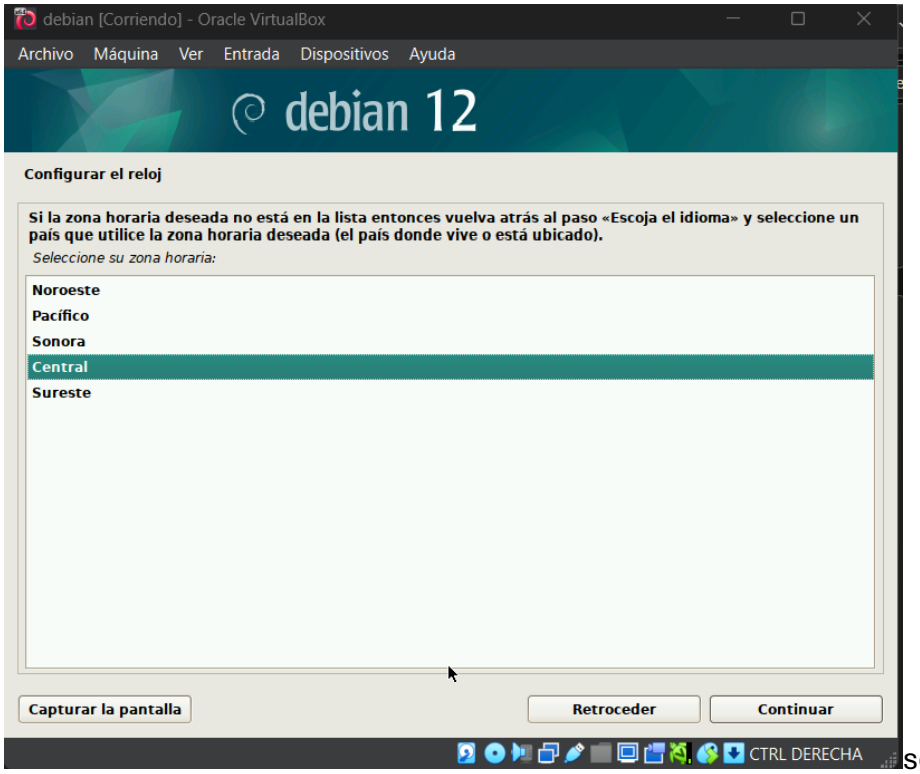
Ingresamos el nombre de usuario



En la contraseña ingresa la misma que ingresaste anteriormente

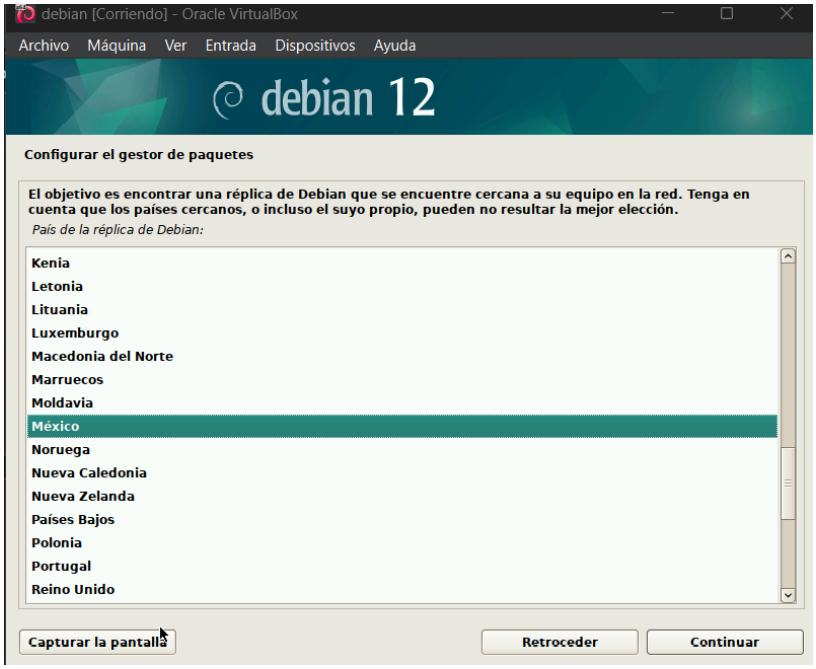


En el reloj seleccionamos central

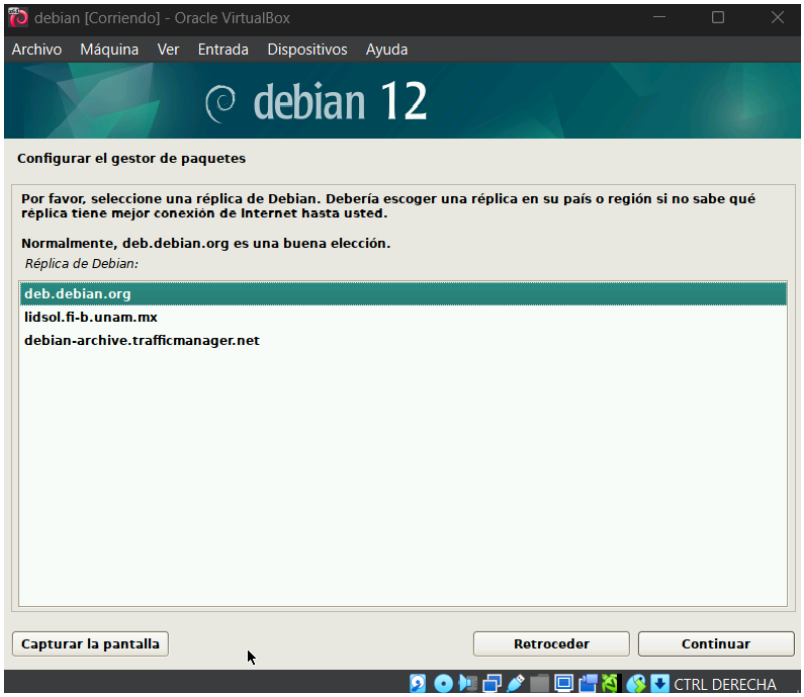


seleccionamos no

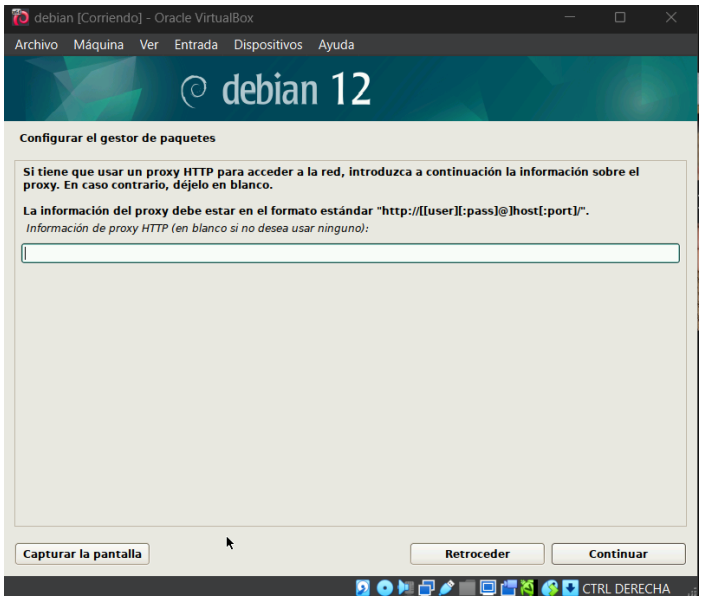
Seleccionamos nuestro pais



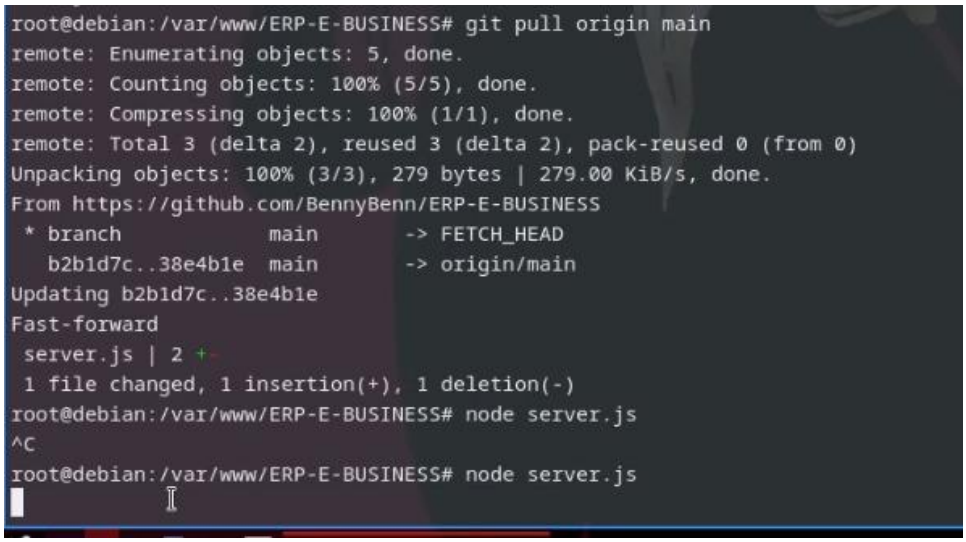
Seleccionamos la primer opcion



La siguiente parte se deja en blanco



Una vez teniendo todo instalado abrimos una terminal y clonamos la repo y corremos el archivo serve.js



```
root@debian:/var/www/ERP-E-BUSINESS# git pull origin main
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 2), reused 3 (delta 2), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 279 bytes | 279.00 KiB/s, done.
From https://github.com/BennyBenn/ERP-E-BUSINESS
 * branch            main       -> FETCH_HEAD
   b2b1d7c..38e4b1e  main       -> origin/main
Updating b2b1d7c..38e4b1e
Fast-forward
 server.js | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
root@debian:/var/www/ERP-E-BUSINESS# node server.js
^C
root@debian:/var/www/ERP-E-BUSINESS# node server.js
```

Estructura de la BD

Nuestra base de datos esta conformada por 9 tablas de las cuales estan relacionadas entre ellas para poder tener un manejo de datos mas controlado y en cascada

Algunas de las relaciones que tenemos son:

MATERIAS → PROFESORES

Un profesor puede impartir varias materias, y una materia puede ser impartida por varios profesores. Esto está vinculado a través de la tabla MATERIASPROFESORES.

PROFESORES → HORARIOS

Un profesor tiene horarios específicos para impartir clases. Cada clase se asigna a un profesor, con el horario correspondiente (lunes, martes, miércoles, jueves y viernes). Esto está relacionado en la tabla HORARIOS.

ALUMNOS → GRUPOALUMNOS

Los alumnos están asociados a grupos específicos en los cuales cursan sus materias. En la tabla GRUPOALUMNOS, cada registro asocia un alumno a una materia y a un grupo específico. Un alumno puede estar inscrito en varios grupos.

ALUMNOS → MATERIAS

Los alumnos están inscritos en las materias, y esas materias pertenecen a un grupo específico, que se asigna a un profesor. Esto es posible a través de la tabla GRUPOALUMNOS.

MATERIAS → GRUPOALUMNOS

Las materias se asignan a grupos de estudiantes en la tabla GRUPOALUMNOS. Los alumnos se asignan a las materias dentro de un grupo, y cada grupo está relacionado con una materia específica.

MATERIASPROFESORES → MATERIAS

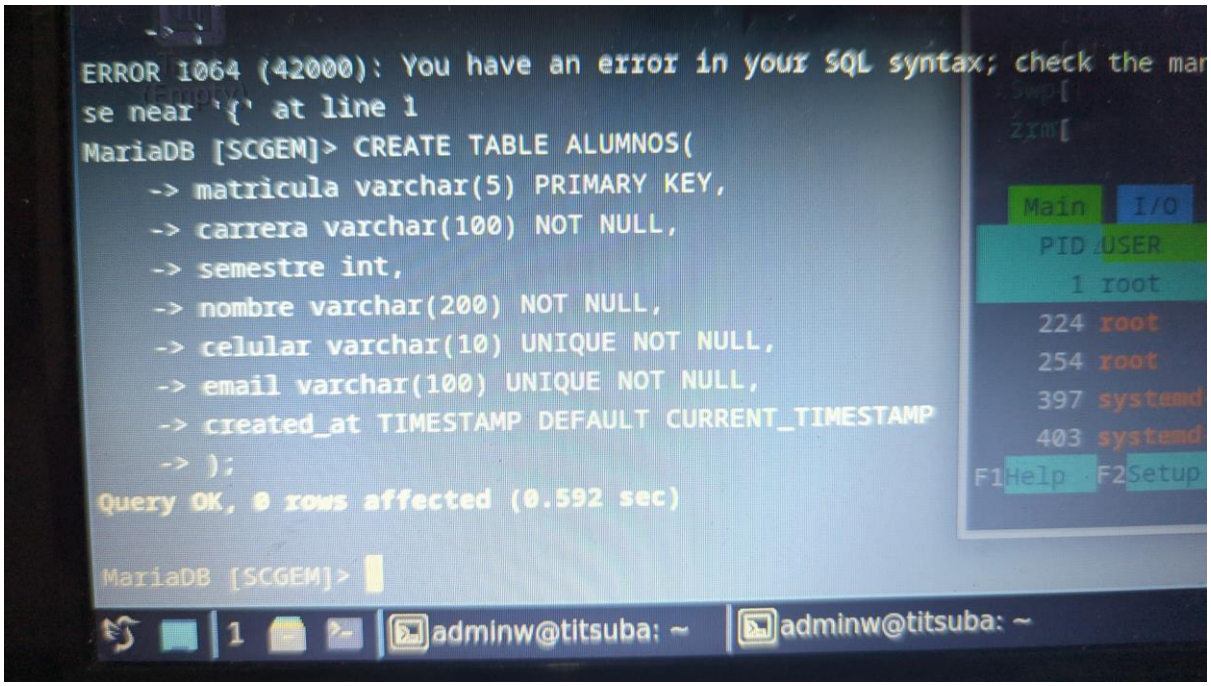
Cada materia es impartida por uno o más profesores, lo que está relacionado a través de la tabla MATERIASPROFESORES.

MATERIASPROFESORES → PROFESORES

Cada grupo de materias es impartido por uno o más profesores. Se asignan en la tabla MATERIASPROFESORES.

Muestra de la declaración de nuestras tablas

Creacion de nuestra tabla alumnos



Creacion de la tabla de profesores

```
MariaDB [SCGEM]> CREATE TABLE PROFESORES(
    -> id_profesor varchar(5) PRIMARY KEY,
    -> nombre varchar(200) NOT NULL,
    -> celular varchar(10) UNIQUE NOT NULL,
    -> email varchar(100) UNIQUE NOT NULL,
    -> created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.235 sec)
```

Creacion de la tabla usuarios

```
MariaDB [SCGEM]> CREATE TABLE USUARIOS(
    -> user_name varchar(30) UNIQUE NOT NULL,
    -> password varchar(30) UNIQUE NOT NULL,
    -> id_user varchar(5) PRIMARY KEY,
    -> user_role varchar(2) NOT NULL
    -> );
Query OK, 0 rows affected (0.210 sec)
```

Creacion de la tabla de calificaciones

```
MariaDB [SCGEM]> CREATE TABLE CALIFICACIONES(
    -> id_materia varchar(5) NOT NULL,
    -> matricula varchar(5) NOT NULL,
    -> calif_p1 varchar(2),
    -> calif_p2 varchar(2),
    -> calif_final varchar(2),
    -> ciclo_cursando varchar(15)
    -> );
Query OK, 0 rows affected (0.190 sec)
```

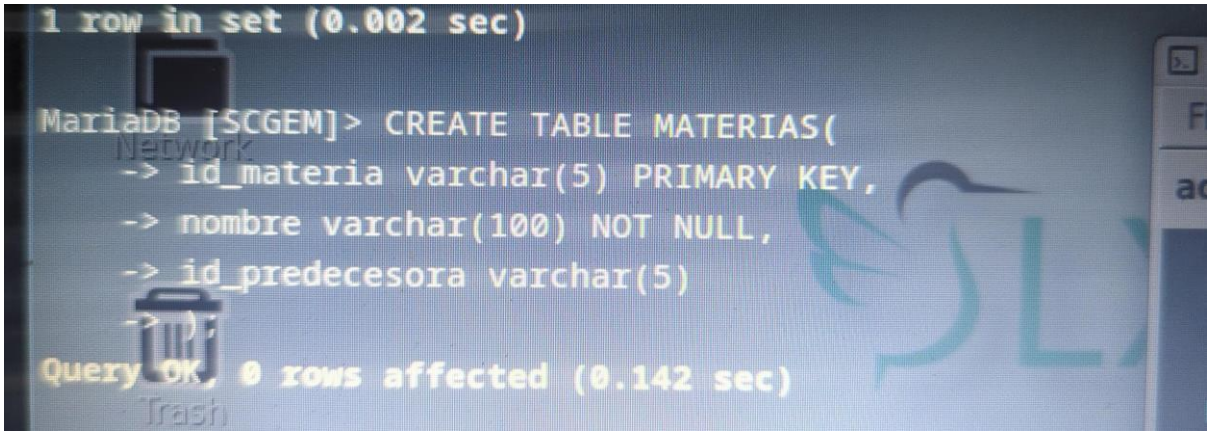
Creacion de la tabla de pagos

```
Query OK, 0 rows affected (0.190 sec)

MariaDB [SCGEM]> CREATE TABLE PAGOS(
    -> matricula varchar(5) NOT NULL,
    -> ciclo_cursando varchar(15) NOT NULL,
    -> pago1 TIMESTAMP,
    -> pago2 TIMESTAMP,
    -> pago3 TIMESTAMP,
    -> pago4 TIMESTAMP,
    -> pago5 TIMESTAMP,
    -> pago6 TIMESTAMP,
    -> pago_mensual varchar(5),
    -> total varchar(10),
    -> status varchar(1)
    -> );
Query OK, 0 rows affected (0.167 sec)

MariaDB [SCGEM]> CREATE TABLE ASISTENCIAS(
    -> matricula varchar(5) not null,
    -> id_materia varchar(5) not null,
    -> codigo_asistencia varchar(10) not null,
    -> asistencia TIMESTAMP
    -> );
```

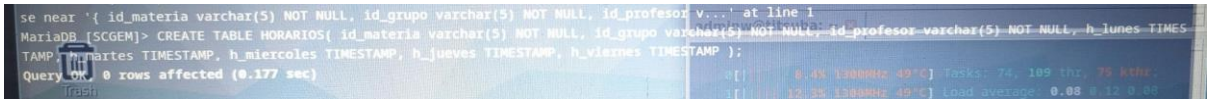
Creacion de la tabla de materiasProfesor

```
| GRUPOALUMNOS     |
| MATERIAS         |
| PROFESORES       |
+------------------+
4 rows in set (0.001 sec)

MariaDB [SCGEM]> CREATE TABLE MATERIASPROFESORES(
    -> id_profesor varchar(5) NOT NULL,
    -> id_grupo varchar(5) NOT NULL
    -> );
Query OK, 0 rows affected (0.181 sec)

MariaDB [SCGEM]>
```

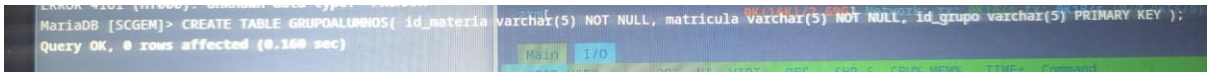
Creacion de la tabla materias



creacion de la tabla horarios



creacion de la tabla grupoALumnos



Codigo

Nuestra app no trabaja con diferentes archivos para priorizar la seguridad, en cambio estamos usando vistas, todo el html esta englobado en un solo archivo “index.html” lo que hacemos para mostrar las diferentes ventanas de usuario es manejar vistas, dependiendo de que es lo que el usuario desee ver, se llamara a el div correspondiente.

Devido a que la app funciona por medio de vistas mostraremos la vista a la que hacemos referecia asi como la explicacion de lo que hace la vista

<!-- Login -->

```
 <div id="login-content" class="content-section">
  <h2>Login</h2>
  <form id="login-form">
   <div class="form-group">
    <label for="username">Usuario</label>
```

```
      <input type="text" id="username" required>
    </div>
    <div class="form-group">
      <label for="password">Contraseña</label>
      <input type="password" id="password" required>
    </div>
    <button type="submit" class="action-button">Login</button>
  </form>
  <div id="error-message"></div>
</div>
```

Este fragmento de código corresponde a la **pantalla de inicio de sesión** de la aplicación.

- **Formulario de inicio de sesión:** Permite a los usuarios ingresar su **nombre de usuario** y **contraseña**.
- **Campos:** Los campos de entrada están marcados como **"Usuario"** y **"Contraseña"**, y ambos son obligatorios para enviar el formulario.
- **Botón de login:** Al hacer clic en el botón **"Login"**, se envían los datos al servidor para verificar las credenciales.

<!----- Panel de estudiantes ----->

```
<div id="student-content" class="content-section">
  <button class="logout-button" onclick="handleLogout()">Cerrar sesion</button>
  <h2>Panel de estudiantes</h2>
  <div class="nav-tabs">
    <div class="nav-tab active" onclick="switchTab('enrolled-subjects',
this)">Materias</div>
    <div class="nav-tab" onclick="switchTab('grades', this)">Calificaciones</div>
    <div class="nav-tab" onclick="switchTab('kardex', this)">Kardez</div>
```

```
  <div class="nav-tab" onclick="switchTab('payments', this)">Pagos</div>
  <div class="nav-tab" onclick="switchTab('attendance', this)">Asistencias</div>
</div>

<div id="enrolled-subjects" class="tab-content active">
  <h3>Materias </h3>
  <table>
    <thead>
      <tr>
        <th>Nombre</th>
        <th>Profesor</th>
        <th>Horarios</th>
        <th>Grupos</th>
      </tr>
    </thead>
    <tbody id="subjects-table">
    </tbody>
  </table>
</div>

<div id="grades" class="tab-content">
  <h3>Calificaciones</h3>
  <table>
    <thead>
```

```
      <tr>
        <th>Nombre</th>
        <th>Parcial 1</th>
        <th>Parcial 2</th>
        <th>Final</th>
        <th>Promedio</th>
      </tr>
    </thead>
    <tbody id="grades-table">
    </tbody>
  </table>
</div>


<div id="kardex" class="tab-content">
  <h3>Historial academico</h3>
  <table>
    <thead>
      <tr>
        <th>Materia</th>
        <th>Periodo</th>
        <th>Calificacion final</th>
        <th>Estado</th>
      </tr>
    </thead>
```

```
    <tbody id="kardex-table">
    </tbody>
  </table>
</div>

<div id="payments" class="tab-content">
  <h3>Estado de pagos</h3>
  <table>
    <thead>
      <tr>
        <th>Mes</th>
        <th>Cantidad</th>
        <th>Fecha de corte</th>
        <th>Estado</th>
        <th>Accion</th>
      </tr>
    </thead>
    <tbody id="payments-table">
    </tbody>
  </table>
</div>

<div id="attendance" class="tab-content">
  <h3>Registro de asistencia</h3>
```

```
    <div class="qr-section">
      <div id="qr-reader"></div>
      <p>Ingresa codigo de asistencia:</p>
      <input type="text" id="attendance-code" placeholder="Enter code">
      <button onclick="submitAttendance()" class="action-button">Submit</button>
    </div>
  </div>
</div>
```

Este código muestra la interfaz del panel de control de un estudiante, donde puede consultar diferentes secciones:

Botón de Cerrar Sesión:

Permite al usuario cerrar sesión con un solo clic.

Navegación por Pestañas:

Hay varias pestañas (tab) que el estudiante puede seleccionar para ver diferentes tipos de información:

Materias: Muestra una lista de las materias en las que está inscrito el estudiante, con el nombre de la materia, el profesor, los horarios y los grupos.

Calificaciones: Muestra las calificaciones del estudiante, divididas por parciales y final, además de su promedio.

Kardex: Muestra el historial académico del estudiante, con las materias pasadas, la calificación final y el estado de cada una.

Pagos: Permite ver el estado de los pagos, incluyendo el mes, la cantidad, la fecha de corte y el estado de cada pago.

Asistencias: Permite al estudiante registrar su asistencia escaneando un código QR o ingresando un código manualmente.

Interacción de Pestañas:

Cada pestaña carga un contenido diferente sin recargar la página. El estudiante puede cambiar entre pestañas sin perder información.

Contenido Dinámico:

Las tablas en cada sección (materias, calificaciones, historial académico, pagos) están vacías al principio y se llenarán dinámicamente con la información correspondiente.

Cada sección tiene su propio conjunto de datos (por ejemplo, materias, pagos, etc.) que se cargan según lo que corresponda al estudiante.

Logout

Panel de profesor

```
 <div id="professor-content" class="content-section">
  <button class="logout-button" onclick="handleLogout()">Logout</button>
  <h2>Panel de profesor</h2>
  <div class="nav-tabs">
   <div class="nav-tab active" onclick="switchTab('teaching-schedule', this)">Horario
de clases</div>
   <div class="nav-tab" onclick="switchTab('attendance-qr', this)">QR de
asistencia</div>
   <div class="nav-tab" onclick="switchTab('grade-entry', this)">Ingreso de
Calificaciones</div>
  </div>
<div id="teaching-schedule" class="tab-content active">
  <h3>Horario de clases</h3>
  <table>
    <thead>
      <tr>
        <th>Materia</th>
        <th>Grupo</th>
        <th>Horario</th>
      </tr>
    </thead>
    <tbody id="schedule-table">
    </tbody>
  </table>
</div>

<div id="attendance-qr" class="tab-content">
  <h3>Generar QR de asistencia</h3>
  <div class="form-group">
    <label>Elegir materia</label>
    <select id="subject-select">
```

```
    </select>
  </div>
  <button onclick="QR_CODE_GEN()" class="action-button">Generar
QR</button>
  <div id="qr-display"></div>
</div>

<div id="grade-entry" class="tab-content">
  <h3>Ingreso de calificaciones</h3>
  <div class="form-group">
    <label>Elija materia</label>
    <select id="grade-subject-select">
    </select>
  </div>
  <table>
    <thead>
      <tr>
        <th>Estudiante</th>
        <th>Parcial 1</th>
        <th>Parcial 2</th>
        <th>Final</th>
        <th>Accion</th>
      </tr>
    </thead>
    <tbody id="grade-entry-table">
    </tbody>
  </table>
</div>
```

Botón de Cerrar Sesión:

Permite al profesor cerrar sesión de la aplicación.

Navegación por Pestañas:

El panel tiene tres pestañas que el profesor puede seleccionar para acceder a diferentes funciones:

Horario de clases: Muestra el horario de clases que el profesor tiene asignadas, con la materia, grupo y horario.

QR de asistencia: Permite generar un código QR para registrar la asistencia de los estudiantes en una clase.

Ingreso de Calificaciones: Permite ingresar las calificaciones de los estudiantes, con campos para los parciales y la calificación final.

Contenido Dinámico en Cada Pestaña:

Horario de clases: Muestra una tabla con las materias, los grupos y los horarios asignados al profesor.

QR de asistencia: Permite al profesor seleccionar una materia y generar un código QR para que los estudiantes lo escaneen y registren su asistencia.

Ingreso de calificaciones: El profesor puede seleccionar una materia y luego ingresar las calificaciones de los estudiantes en una tabla, donde puede introducir las notas de los parciales y la calificación final.

<!----- Panel de administrativos ----->

```
  <div id="admin-staff-content" class="content-section">
    <button class="logout-button"
onclick="handleLogout()">Logout</button>
    <h2>Panel de administrativos</h2>
    <div class="nav-tabs">

      <div class="nav-tab active" onclick="switchTab('student-
management', this)">Estudiantes</div>
      <div class="nav-tab" onclick="switchTab('professor-
management', this)">Profesores</div>
      <div class="nav-tab" onclick="switchTab('subject-management',
this)">Materias</div>

      <!--
        <div id="student-management" class="tab-content
active"></div>
        <div id="professor-management" class="tab-content"></div>
        <div id="subject-management" class="tab-content"></div>
-->
      <div class="nav-tab" onclick="switchTab('reports',
this)">Reportes</div>
    </div>

  </div>
```

Este código representa la interfaz del panel de control para los administrativos, quienes tienen acceso a funciones clave para gestionar a estudiantes, profesores, materias y generar reportes.

Botón de Cerrar Sesión:

Permite a los administrativos cerrar sesión de la aplicación con un solo clic.

Navegación por Pestañas:

Hay varias pestañas (tab) que los administrativos pueden seleccionar para acceder a diferentes secciones:

Estudiantes: Permite gestionar la información de los estudiantes.

Profesores: Permite gestionar la información de los profesores.

Materias: Permite gestionar las materias ofrecidas en el sistema.

Reportes: Permite generar o ver reportes relacionados con los estudiantes, profesores o cualquier otra área administrativa.

Contenido Dinámico en Cada Pestaña:

Cada pestaña permitirá al administrador acceder a las herramientas necesarias para gestionar estudiantes, profesores o materias, y para ver o generar reportes, según la pestaña seleccionada.

<!----- Panel de linuxero maestro ----->

```
 <div id="super-admin-content" class="content-section">

   <button class="logout-button" onclick="handleLogout()">Logout</button>

   <h2>Administracion del sistema</h2>

   <div class="nav-tabs">


    <div class="nav-tab active" onclick="switchTab('user-management',
this)">Usuarios</div>

    <div class="nav-tab" onclick="switchTab('system-config',
this)">Configuraciones</div>

    <div class="nav-tab" onclick="switchTab('system-logs', this)">Registros</div>


     </div>
```

Botón de Cerrar Sesión:

Permite al superadministrador cerrar sesión de la aplicación.

Navegación por Pestañas:

Hay tres pestañas disponibles para gestionar distintas áreas del sistema:

Usuarios: Permite gestionar los usuarios del sistema, como crear, editar o eliminar usuarios.

Configuraciones: Permite modificar las configuraciones del sistema, como parámetros del servidor o ajustes globales de la aplicación.

Registros: Muestra los registros del sistema, permitiendo al superadministrador ver eventos, errores y otras actividades que han ocurrido en el sistema.

Contenido Dinámico en Cada Pestaña:

Al hacer clic en cada pestaña, se cargará el contenido correspondiente, permitiendo gestionar usuarios, hacer configuraciones o revisar registros del sistema.

Logica JS del archivo index.html

Roles

```
const ROLES = {
  STUDENT: 1,
  PROFESSOR: 2,
  ADMIN_STAFF: 3,
  SUPER_ADMIN: 99
};
```

ROLES es un objeto que contiene cuatro propiedades, cada una correspondiente a un rol de usuario en el sistema.

STUDENT: 1: El rol de estudiante está representado por el valor 1.

PROFESSOR: 2: El rol de profesor tiene el valor 2.

ADMIN_STAFF: 3: El rol de personal administrativo está representado por el valor 3.

SUPER_ADMIN: 99: El superadministrador, quien tiene acceso completo al sistema, está representado por el valor 99.

Este objeto ROLES probablemente se usa en otras partes del código para comprobar qué tipo de usuario ha iniciado sesión y asignarles acceso a las diferentes funcionalidades del sistema según su rol.

Estilos tabla dinámica

```
const dataTableStyle = {
pageLength: 5,
lengthMenu: [5, 10, 20],
destroy: true, language: {
lengthMenu: "Mostrar MENU registros por pagina",
zeroRecords: "No se encontraron registros",
info: "Mostrando pagina PAGE de PAGES",
infoEmpty: "No hay registros disponibles",
infoFiltered: "(filtrado de MAX registros totales)",
search: "Buscar:"
, paginate: {
first: "Primero",
last: "Ultimo",
next: "Siguiente",
previous: "Anterior"
}
}
```

};

pageLength: 5:

Esto define la cantidad de registros que se mostrarán por página de la tabla. En este caso, serán 5 registros por página.

lengthMenu: [5, 10, 20]:

Esto proporciona un menú para que el usuario pueda elegir cuántos registros quiere ver por página. En este caso, las opciones son 5, 10 o 20 registros por página.

destroy: true:

Esta opción permite que la tabla sea destruida y reconstruida si ya existe. Esto es útil cuando se aplica una nueva configuración o se actualizan los datos de la tabla.

language:

Dentro de este objeto se definen las opciones de idioma para la tabla. Aquí se configuran los textos que se mostrarán al usuario en español.

lengthMenu: Texto que aparece en el menú para elegir la cantidad de registros a mostrar. En español: "Mostrar MENU registros por página".

zeroRecords: Mensaje que aparece cuando no se encuentran registros. En español: "No se encontraron registros".

info: Muestra información sobre la página actual. En español: "Mostrando página PAGE de PAGES".

infoEmpty: Mensaje cuando no hay registros disponibles. En español: "No hay registros disponibles".

infoFiltered: Mensaje que aparece cuando se filtran los registros. En español: "(filtrado de MAX registros totales)".

search: Texto que aparece junto al cuadro de búsqueda. En español: "Buscar:".

paginate: Configuración de los botones de paginación:

first: Texto para el botón que va a la primera página. "Primero".

last: Texto para el botón que va a la última página. "Último".

next: Texto para el botón que va a la siguiente página. "Siguiente".

previous: Texto para el botón que va a la página anterior. "Anterior".

function getBackendIP() {

```
let backendIP = localStorage.getItem('backendIP');


//if (!backendIP) {
  backendIP = prompt("Enter backend IP (e.g., 192.168.18.X):", "192.168.18.4");
  if (backendIP) {
    localStorage.setItem('backendIP', backendIP);
  } else {
    alert("Backend IP is required!");
    return null;
  }
```

Intenta obtener la IP del servidor (backend) desde el almacenamiento local del navegador.

Si no encuentra la IP, le pide al usuario que la ingrese a través de un cuadro de diálogo.

Si el usuario ingresa una IP, la guarda en el almacenamiento local para futuras veces.

Si el usuario no ingresa ninguna IP, muestra un mensaje de error y devuelve null.

cambia paginas

```
function switchTab(tabId, element) {
  // Remove active class from all tabs
  document.querySelectorAll('.nav-tab').forEach(tab => {
tab.classList.remove('active'); });
  document.querySelectorAll('.tab-content').forEach(content => {
content.classList.remove('active'); });


  // establece clase activa a la pagina actual
  element.classList.add('active');
  document.getElementById(tabId).classList.add('active');
```

```
    if (tabId === 'enrolled-subjects') {
      loadSubjects();
    }
    if (tabId === 'grades') {
      loadGrades();
    }
    if (tabId === 'kardex') {
      loadKardez();
    }
    if (tabId === 'payments') {
      loadPagos();
    }
    if (tabId === 'teaching-schedule') {
      loadSubjectsProf();
    }
  }
```

Esta función cambia entre diferentes páginas o secciones dentro de la interfaz de usuario de la aplicación.

Desactiva las pestañas y contenidos anteriores:

Elimina la clase active de todas las pestañas (nav-tab) y secciones de contenido (tab-content) para que no se muestren.

Añade la clase active a la pestaña seleccionada y su contenido correspondiente para mostrarlo.

Dependiendo de la pestaña (como Materias, Calificaciones, Kardex, etc.), se llama a una función específica para cargar la información:

loadSubjects() para Materias.

loadGrades() para Calificaciones.

loadKardez() para Kardex.

loadPagos() para Pagos.

loadSubjectsProf() para el horario de clases del profesor.

// Login hanlder

```
document.getElementById('login-form').addEventListener('submit', async (e) => {
  e.preventDefault();

  const username = document.getElementById('username').value;
  const password = document.getElementById('password').value;
  const errorMessage = document.getElementById('error-message');
  ///Cambio solo las ip, no el puerto
  try {
    let ip = '192.168.18.27';
    console.log(backendIP);
    const response = await fetch('http://192.168.18.9:3000/login', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ username, password })
    });

    const data = await response.json();

    if (response.ok) {
      localStorage.setItem('authToken', data.token);
      errorMessage.textContent = '';
      showUserRole(data.user_role);
    } else {
      errorMessage.textContent = data.message || 'Login failed';
    }
  } catch (error) {
```

```
    errorMessage.textContent = 'Connection error. Please try again.';
    console.error('Login error:', error);
  }

  console.log("Token being sent:", localStorage.getItem('authToken'));

});
```

Este código gestiona el inicio de sesión del usuario al enviar el formulario de login.

Cuando el formulario de login se envía, se previene que la página se recargue (e.preventDefault()).

Obtiene el nombre de usuario (username) y la contraseña (password) que el usuario ingresó en el formulario.

El código realiza una solicitud POST al servidor de la IP 192.168.18.9:3000/login (se conecta al servidor para autenticar al usuario).

Se envía el nombre de usuario y la contraseña en formato JSON.

Si la respuesta del servidor es exitosa (response.ok), guarda el token de autenticación en el almacenamiento local (localStorage).

Muestra el rol del usuario usando la función showUserRole().

Si el login falla, muestra un mensaje de error, que proviene de la respuesta del servidor.

Si hay un error al intentar conectarse con el servidor (por ejemplo, problemas de red), se muestra un mensaje de error y se registra el error en la consola.

// display por roles

```
function showUserRole(role) {
```

```
    document.querySelectorAll('.content-section').forEach(section => {
section.style.display = 'none'; });


    switch (parseInt(role)) {
      case ROLES.SUPER_ADMIN:
        document.getElementById('super-admin-content').style.display = 'block';
        break;
      case ROLES.PROFESSOR:
        document.getElementById('professor-content').style.display = 'block';
        break;
      case ROLES.STUDENT:
        document.getElementById('student-content').style.display = 'block';
        break;
      case ROLES.ADMIN_STAFF:
        document.getElementById('admin-staff-content').style.display = 'block';
        break;
      default:
        document.getElementById('login-content').style.display = 'block';
    }
  }
```

Primero, se ocultan todas las secciones de contenido (content-section) estableciendo su estilo display a 'none'.

Utiliza un switch para verificar el rol del usuario (que se pasa como parámetro role).

Según el valor del rol, se muestra la sección correspondiente:

SUPER_ADMIN: Muestra la sección de administración del sistema (super-admin-content).

PROFESSOR: Muestra el panel del profesor (professor-content).

STUDENT: Muestra el panel del estudiante (student-content).

ADMIN_STAFF: Muestra el panel administrativo (admin-staff-content).Si el rol no coincide con ninguno de los roles predefinidos, se muestra la sección de login (login-content).

CODIGO ESTUDIANTES

```
////////////////      MATERIAS      ////////////////////////////////////////////
async function loadSubjects() {
  const token = localStorage.getItem("authToken");
  const response = await fetch("http://192.168.18.9:3000/student/tabla-datos-estudiante/", { headers: { "Authorization": `Bearer ${token}` } });
  console.log("Token:", localStorage.getItem("authToken"));

  if (!response.ok) {
    console.error("Error al obtener las materias");
    return;
  }

  const subjects = await response.json();
  const tbody = document.getElementById("subjects-table");
  tbody.innerHTML = "";

  subjects.forEach(subject => {
    const row = `<tr>
    <td>${subject.materia_nombre}</td>
    <td>${subject.profesor_nombre}</td>
    <td>${subject.horarios.replace(/\n/g, '<br>')}</td> <!-- Replace \n with <br> -->
```

```
    <td>${subject.id_grupo}</td>
  </tr>`;
    tbody.innerHTML += row;
  });
}

////////////////      CALIFICACIONES      ////////////////////////////////////////////
async function loadGrades() {
  const token = localStorage.getItem("authToken");
  const response = await fetch("http://192.168.18.9:3000/student/tabla-
calificaciones/", { headers: { "Authorization": `Bearer ${token}` } });;

  if (!response.ok) {
    console.error("Error al obtener las materias");
    return;
  }

  const subjects = await response.json();
  const tbody = document.getElementById("grades-table");
  tbody.innerHTML = "";

  subjects.forEach(subject => {
    const row = `<tr>
    <td>${subject.materia}</td>
    <td>${subject.calif_p1}</td>
    <td>${subject.calif_p2}</td>
    <td>${subject.calif_final}</td>
    <td>${subject.promedio}</td>
```

```
  </tr>`;
    tbody.innerHTML += row;
  });
  }

////////////////      KARDEZ      ////////////////////////////////////////////
async function loadKardez() {
  const token = localStorage.getItem("authToken");
  const response = await fetch("http://192.168.18.9:3000/student/tabla-kardez/", {
headers: { "Authorization": `Bearer ${token}` } });;

  if (!response.ok) { console.error("Error al obtener las materias");
    return;
  }

  const subjects = await response.json();
  const tbody = document.getElementById("kardex-table");
  tbody.innerHTML = "";

  subjects.forEach(subject => {
  const row = `<tr>
    <td>${subject.materia}</td>
    <td>${subject.periodo}</td>
    <td>${subject.calif_final}</td>
    <td>${subject.estado}</td>
  </tr>`;
  tbody.innerHTML += row;
});
```

```
}

////////////////      PAGOS      ////////////////////////////////////////////

async function loadPagos() {

   const token = localStorage.getItem("authToken");

   const response = await fetch("http://192.168.18.9:3000/student/tabla-pagos/", {
headers: { "Authorization": `Bearer ${token}` } });;


   if (!response.ok) { console.error("Error al obtener las materias");

      return;

   }


   const subjects = await response.json();

   const tbody = document.getElementById("payments-table");

   tbody.innerHTML = "";


   subjects.forEach(subject => {

   const row = `<tr>

      <td>${subject.MES}</td>

      <td>${subject.CANTIDAD}</td>

      <td>${subject.FECHA_CORTE}</td>

      <td>${subject.ESTADO}</td>

      <td>${subject.ACCION}</td>

   </tr>`;

   tbody.innerHTML += row;

});

}
```

```
////////////////      ASISTENCIAS      ////////////////////////////////////////////

function submitAttendance() {
  const token = localStorage.getItem("authToken");
  const attendanceCode = document.getElementById("attendance-code").value.trim();

  console.log("Attendance Code (Front):", attendanceCode); // Debugging

  if (!attendanceCode) {
    alert("Por favor, ingresa un código de asistencia.");
    return;
  }

  fetch("http://192.168.18.9:3000/student/registro-asistencias/", {
    method: "POST",
    headers: {
      "Authorization": `Bearer ${token}`,
      "Content-Type": "application/json" // ✅ Required for JSON body parsing
    },
    body: JSON.stringify({ codigo_asistencia: attendanceCode })
  })
  .then(response => response.json())
  .then(data => {
    if (data.success) {
      alert("Asistencia registrada exitosamente.");
    } else {
      alert("Error al registrar asistencia: " + data.message);
    }
```

```
  })
  .catch(error => {
    console.log("Error:", error);
    alert("Error al conectar con el servidor.");
  });
}
```

loadSubjects():

Obtiene las materias del estudiante desde el servidor y las muestra en una tabla.

Para cada materia, se muestra el nombre, profesor, horarios y grupo.

loadGrades():

Obtiene las calificaciones del estudiante desde el servidor y las muestra en una tabla.

Para cada materia, se muestran las calificaciones de los parciales y el promedio.

loadKardez():

Obtiene el historial académico (kardex) del estudiante y lo muestra en una tabla.

Muestra la materia, periodo, calificación final y el estado (aprobado/reprobado).

loadPagos():

Obtiene el estado de pagos del estudiante y lo muestra en una tabla.

Muestra el mes, cantidad a pagar, fecha de corte, estado del pago y la acción correspondiente.

submitAttendance():

Permite al estudiante registrar su asistencia introduciendo un código de asistencia.

El código se envía al servidor para su validación y registro.

Si se registra correctamente, muestra un mensaje de éxito. Si ocurre un error, muestra un mensaje de error.

CODIGO PROFESORES

```
async function loadSubjectsProf() { const token = localStorage.getItem("authToken");
const response = await fetch("http://192.168.18.9:3000/professor/schedule/", {
```

headers: { "Authorization": Bearer ${token} }}); console.log("Token:", localStorage.getItem("authToken"));

```
 if (!response.ok) {
    console.error("Error al obtener las materias");
    return;
  }

  const subjects = await response.json();
  console.log("retorno",subjects);
  const tbody = document.getElementById("schedule-table");
  tbody.innerHTML = "";

  subjects.forEach(subject => {
    const row = `<tr>
    <td>${subject.materia_nombre}</td>
    <td>${subject.id_grupo}</td>
    <td>${subject.horarios.replace(/\n/g, '<br>')}</td> <!-- Replace
\n with <br> -->
</tr>`;
    tbody.innerHTML += row;
  });
}


////////////////////////////////////////////QR//////////////////////
//////////
async function QR_CODE_GEN() {
  const token = localStorage.getItem("authToken");
  const response = await
fetch("http://192.168.18.9:3000/professor/QR_CODE_GEN/", { headers:
{ "Authorization": `Bearer ${token}` } });
  console.log("Token:", localStorage.getItem("authToken"));

  if (!response.ok) {
    console.error("Error al obtener las materias");
    return;
  }

  const subjects = await response.json();
  console.log("retorno",subjects);
  const tbody = document.getElementById("schedule-table");
```

```
  tbody.innerHTML = "";

  subjects.forEach(subject => {
    const row = `<tr>
    <td>${subject.materia_nombre}</td>
    <td>${subject.id_grupo}</td>
    <td>${subject.horarios.replace(/\n/g, '<br>')}</td> <!-- Replace
\n with <br> -->
</tr>`;
    tbody.innerHTML += row;
```

loadSubjectsProf():

Obtiene las materias y horarios que el profesor está impartiendo desde el servidor.

Los datos recibidos incluyen el nombre de la materia, el grupo y los horarios. Estos se muestran en una tabla.

Se asegura de incluir los saltos de línea (\n) en los horarios al mostrarlos en formato HTML (<br>).

QR_CODE_GEN():

Genera un código QR para la asistencia, basándose en las materias del profesor.

Hace una petición al servidor para obtener las materias que se están enseñando, y se muestra la información de cada una en una tabla (nombre de la materia, grupo, y horarios).

Aunque en el código se llama QR_CODE_GEN(), parece que aún no está implementada la parte de mostrar o generar el QR de asistencia visualmente. Lo que hace actualmente es mostrar los datos de la materia en la tabla.