



Base de datos

Evidencias del proyecto

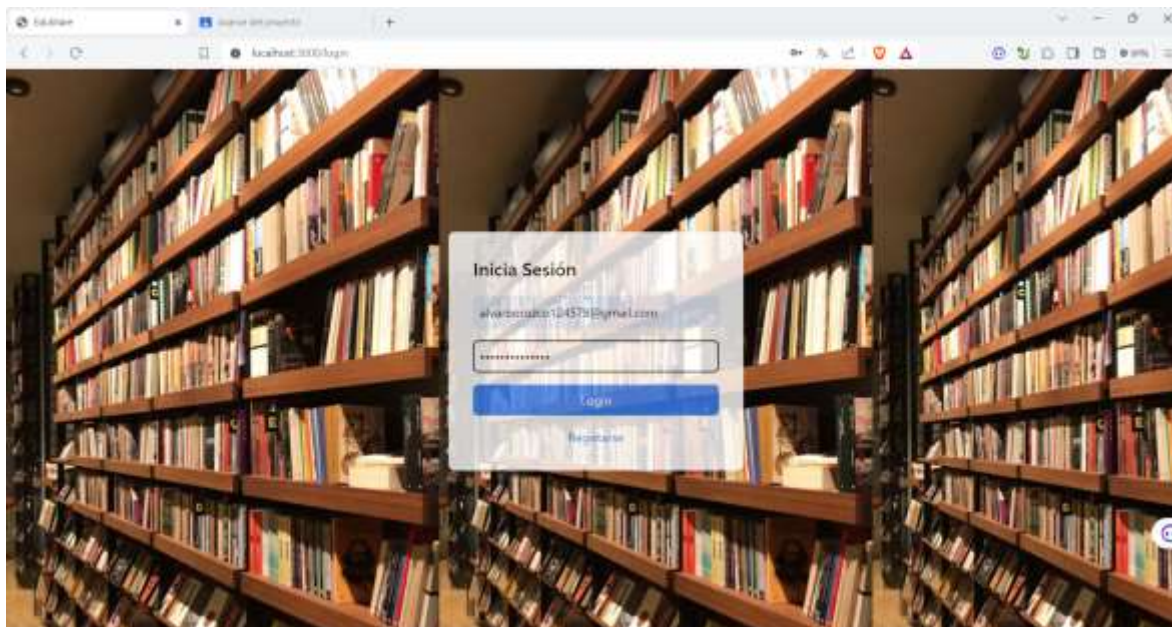
Nombre: Alvaro Orozco Pérez

Carrera: ISIC 6°

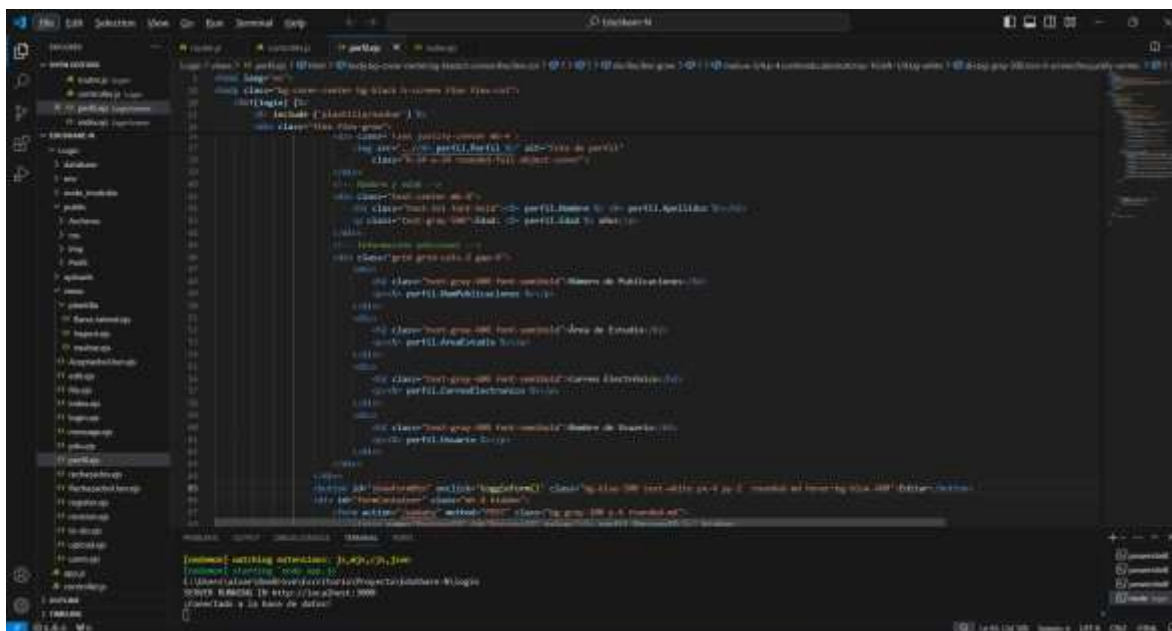
Nombre De la Maestra: Roció Del Carmen Parra Torres

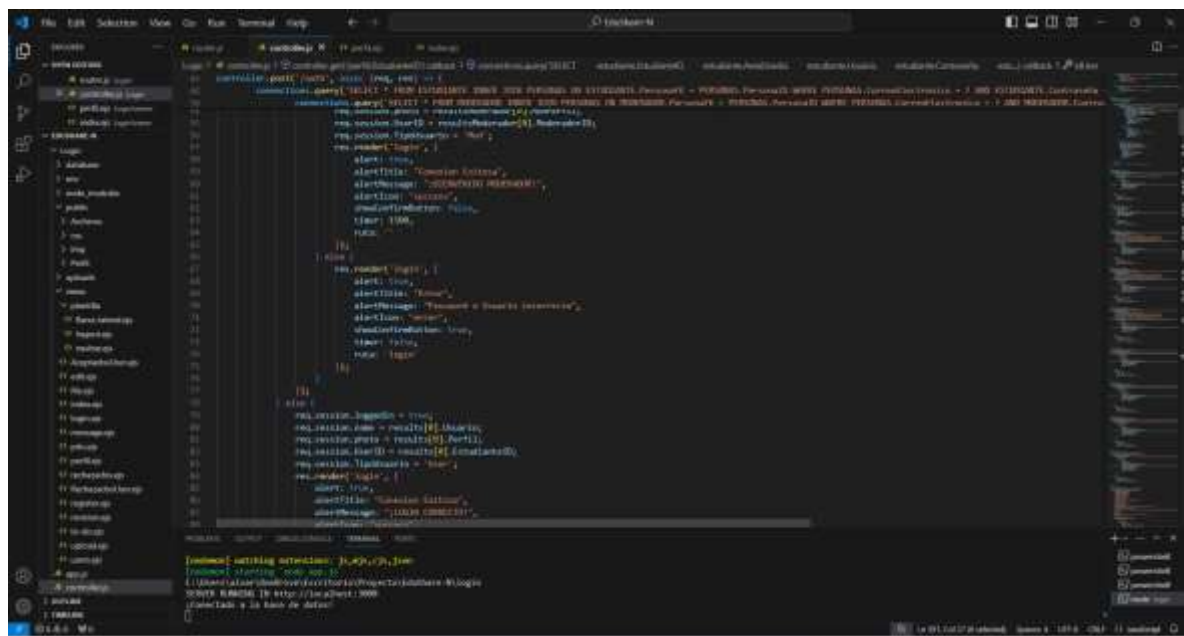
Login

Es el sistema de autenticación muy sencillo, lo que hacemos es una búsqueda rápida con las credenciales del usuario que quiere iniciar sesión, y los datos importantes los guardamos en variables de sesión



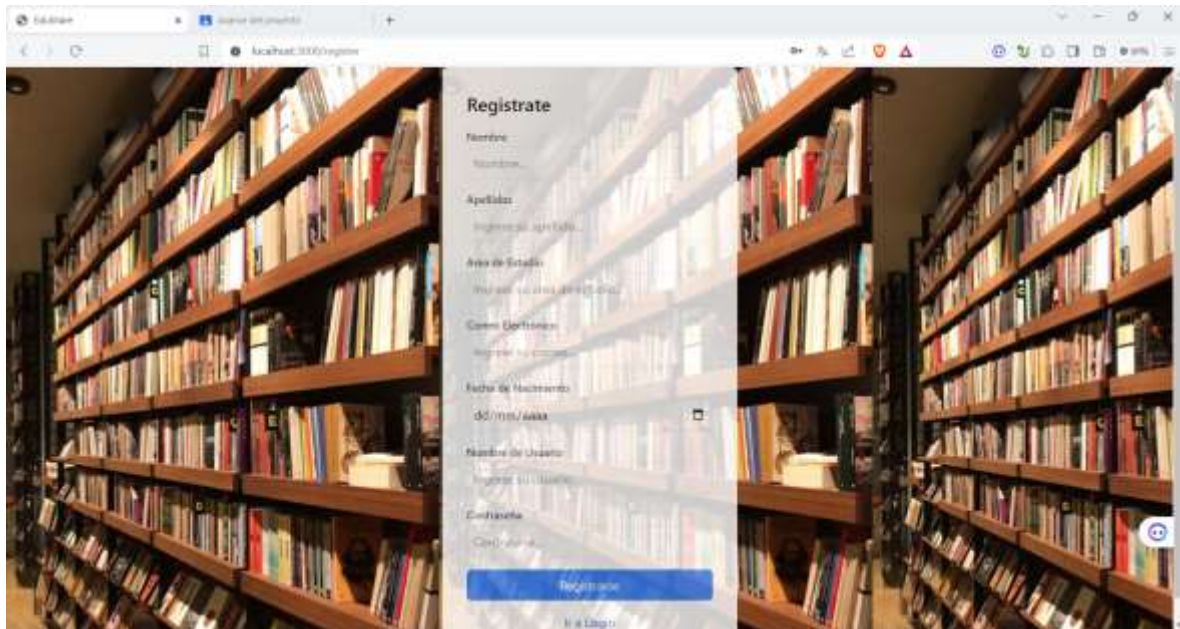
Codigo: Aquí pues esta todo el diseño de la ventana, el formulario y todo el frond end, después esta el controlador, que a grandes rasgos es la consulta



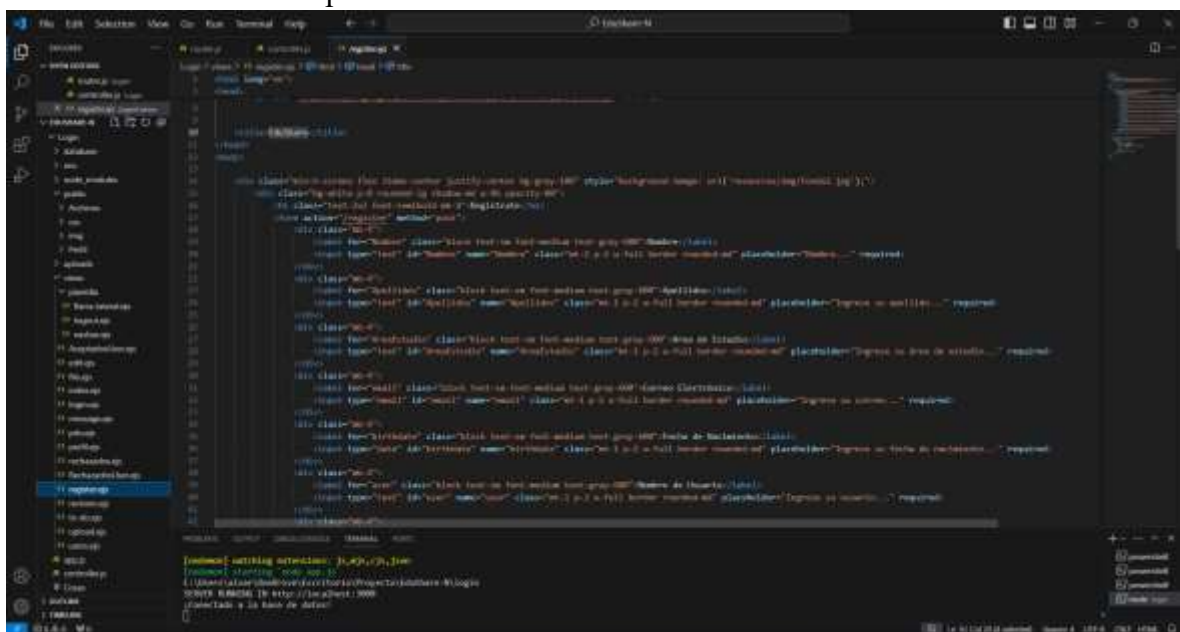


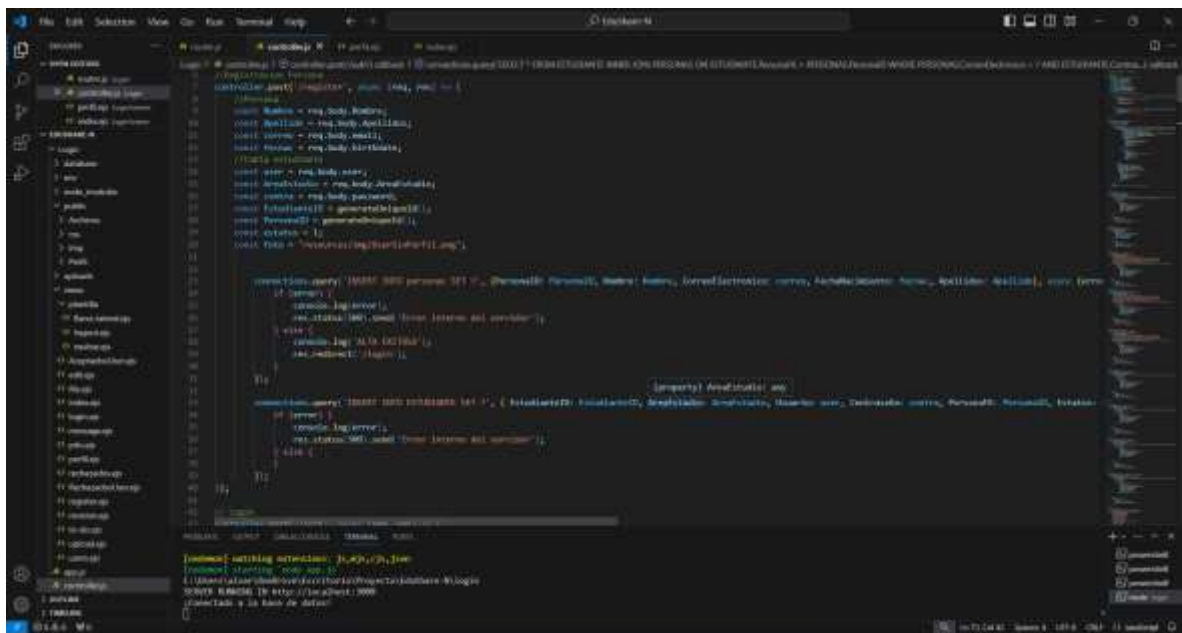
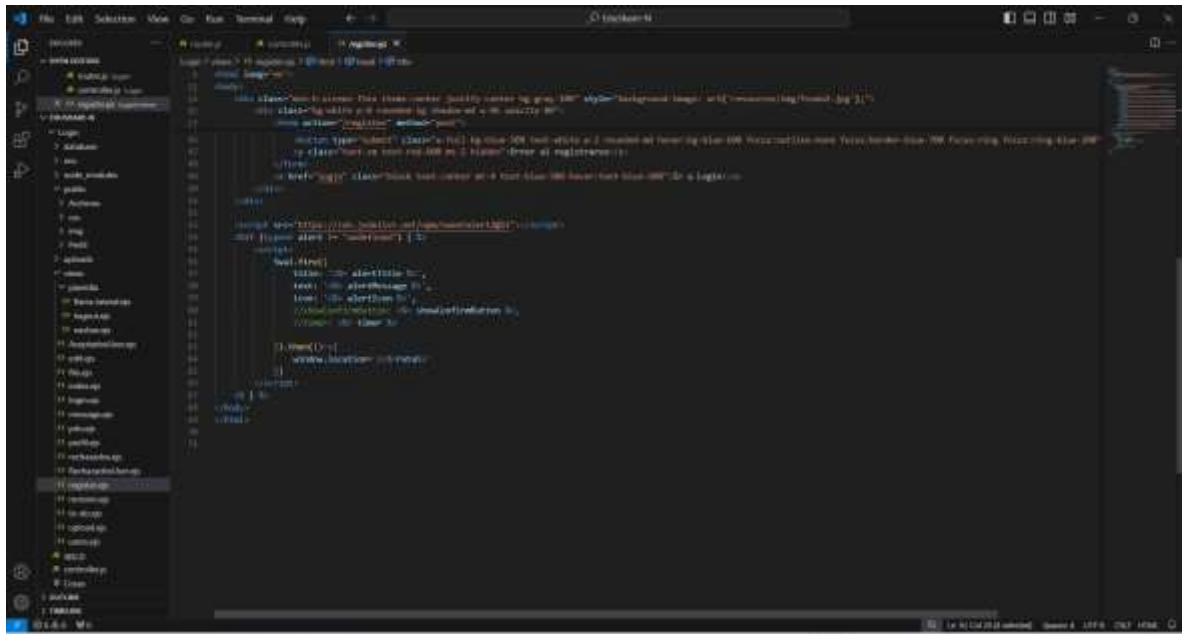
Registro

Aquí se podrán registrar los nuevos usuarios, como tal es un formulario para agarrar los datos de los usuarios



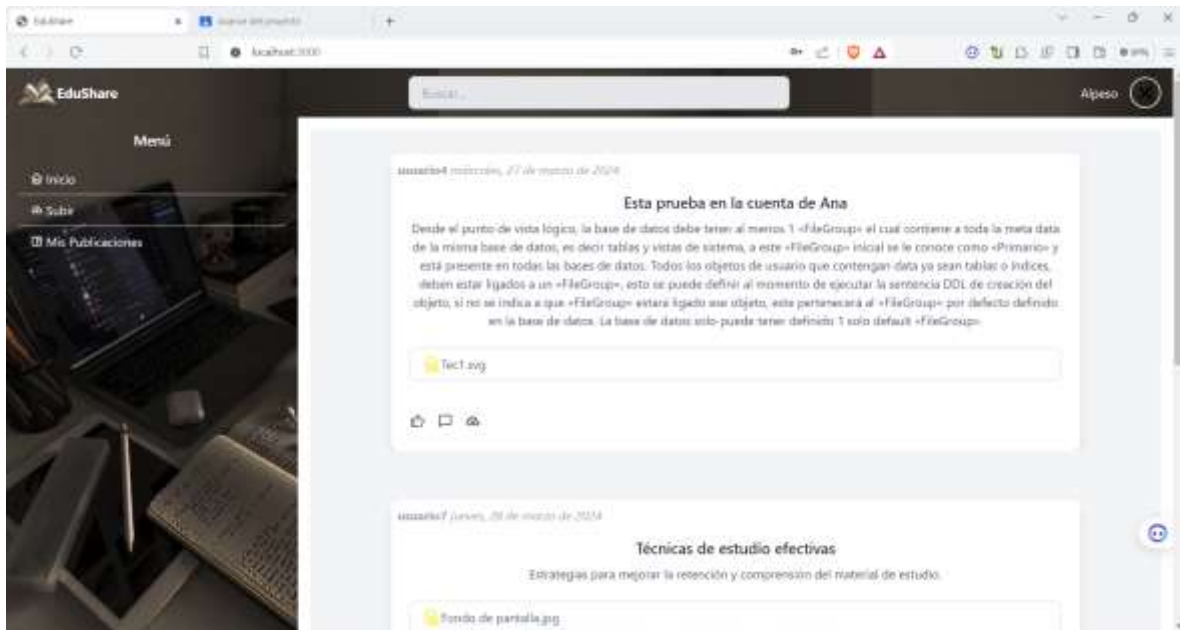
Código: Aquí está el formulario como tal el código, para que visualmente se vea mejor, después está el controlador que este como tal es el INSERT para insertar los datos en la base de datos de manera permanente



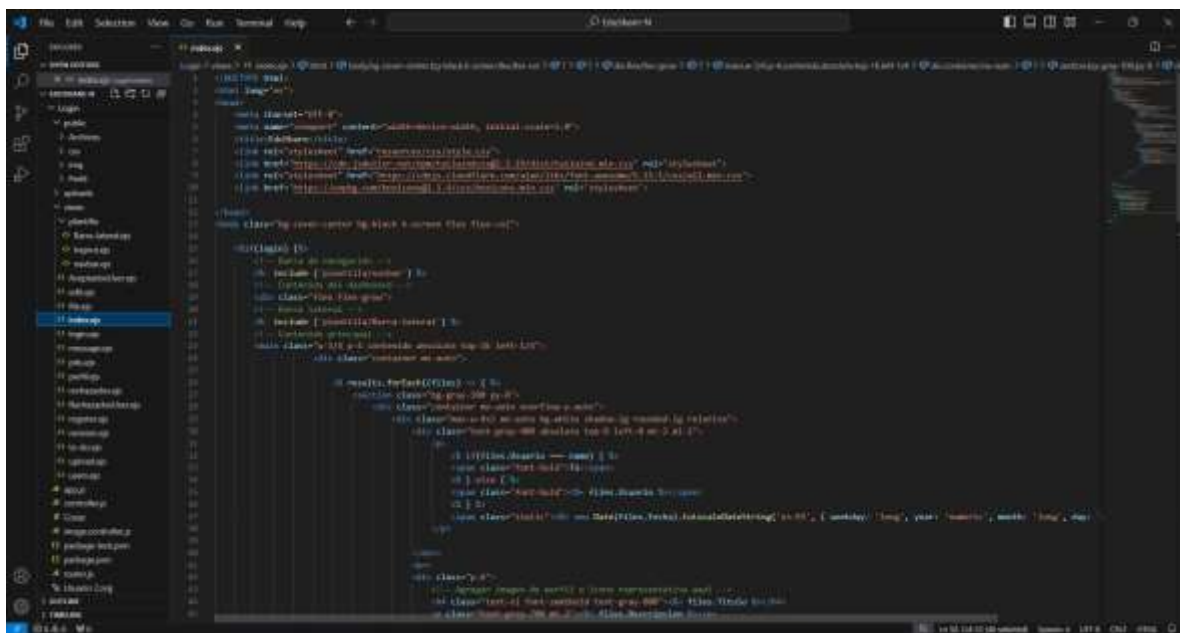


Inicio

Aquí esta como el Home o el feedback, tratando de imitar un poco como lo que hay en Facebook, esto para ver la información de mejor manera, poderlo descargar, comentar y mandar mensajes (lo de los mensajes y comentarios todavía no lo termino)

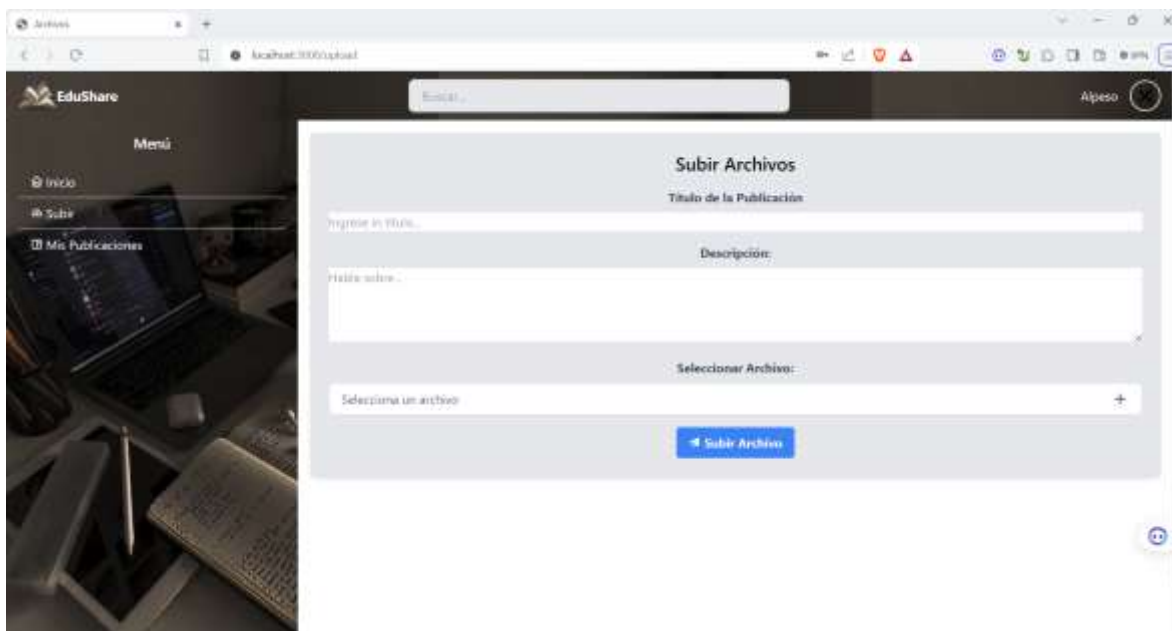


Código: Aquí está todo en código que básicamente esta es una etiqueta selection esta en un bucle determinada por el número de respuestas que tenga la base de datos, lo demás es diseño. En el controlador es esta la consulta en la base de datos y guardada en arreglo para mostrar la información

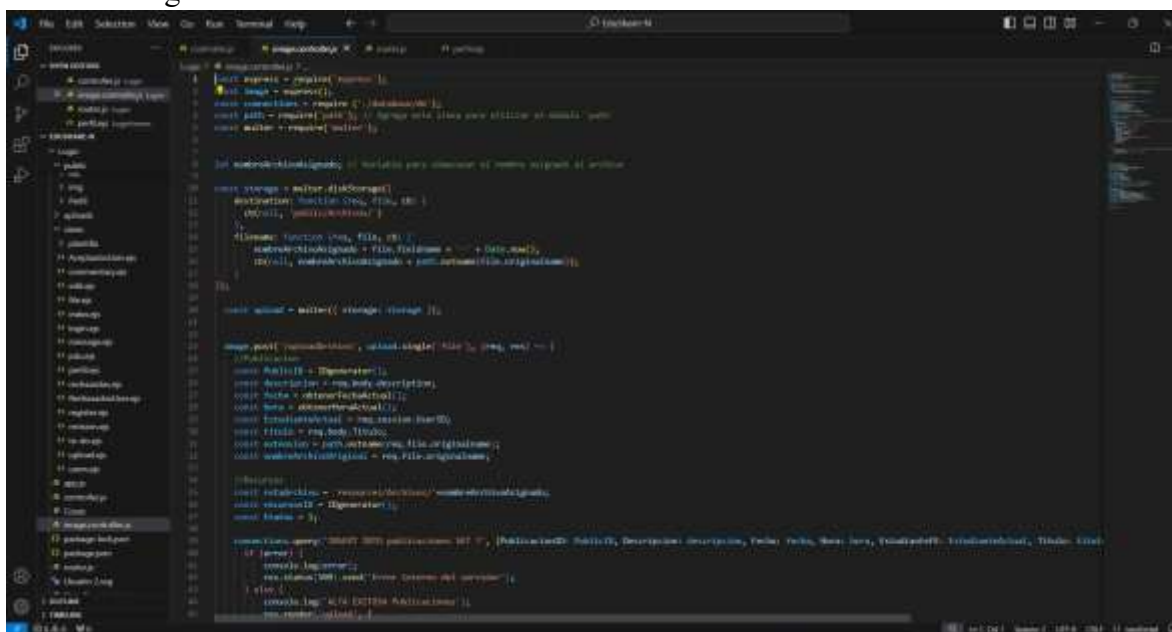


Subir archivos


Este es el formulario para subir archivos a la plataforma, es un diseño muy básico en el que podemos subir la información necesaria y además es muy intuitivo que se debe de subir



Código: En esta parte está el insert, pero también utilice la librería “multer” para subir toda clase de archivos y que fuera compatible con ellos, lógica es muy sencilla sin embargo la librería es algo tediosa de



utilizar



```

// index.js
// @author: [Your Name]
// @version: 1.0.0
// @description: A JavaScript class for a person.

class Person {
  constructor(
    name,
    age,
    gender,
    email,
    password,
    phone,
    address,
    city,
    state,
    zip,
    country,
    occupation,
    education,
    maritalStatus,
    children,
    pets
  ) {
    this.name = name;
    this.age = age;
    this.gender = gender;
    this.email = email;
    this.password = password;
    this.phone = phone;
    this.address = address;
    this.city = city;
    this.state = state;
    this.zip = zip;
    this.country = country;
    this.occupation = occupation;
    this.education = education;
    this.maritalStatus = maritalStatus;
    this.children = children;
    this.pets = pets;
  }

  login() {
    console.log('Login successful');
  }

  logout() {
    console.log('Logout successful');
  }

  updateProfile() {
    console.log('Update profile successful');
  }

  updatePassword() {
    console.log('Update password successful');
  }

  updateEmail() {
    console.log('Update email successful');
  }

  updatePhone() {
    console.log('Update phone successful');
  }

  updateAddress() {
    console.log('Update address successful');
  }

  updateCity() {
    console.log('Update city successful');
  }

  updateState() {
    console.log('Update state successful');
  }

  updateZip() {
    console.log('Update zip successful');
  }

  updateCountry() {
    console.log('Update country successful');
  }

  updateOccupation() {
    console.log('Update occupation successful');
  }

  updateEducation() {
    console.log('Update education successful');
  }

  updateMaritalStatus() {
    console.log('Update marital status successful');
  }

  updateChildren() {
    console.log('Update children successful');
  }

  updatePets() {
    console.log('Update pets successful');
  }
}

// Create a new person object
const person = new Person(
  'John Doe',
  30,
  'Male',
  'john.doe@example.com',
  'password123',
  '123-456-7890',
  '123 Main St',
  'New York',
  'NY',
  '10001',
  'USA',
  'Software Engineer',
  'Bachelor's Degree',
  'Married',
  2,
  ['Dog', 'Cat']
);

// Call the login method
person.login();

// Call the logout method
person.logout();

// Call the updateProfile method
person.updateProfile();

// Call the updatePassword method
person.updatePassword();

// Call the updateEmail method
person.updateEmail();

// Call the updatePhone method
person.updatePhone();

// Call the updateAddress method
person.updateAddress();

// Call the updateCity method
person.updateCity();

// Call the updateState method
person.updateState();

// Call the updateZip method
person.updateZip();

// Call the updateCountry method
person.updateCountry();

// Call the updateOccupation method
person.updateOccupation();

// Call the updateEducation method
person.updateEducation();

// Call the updateMaritalStatus method
person.updateMaritalStatus();

// Call the updateChildren method
person.updateChildren();

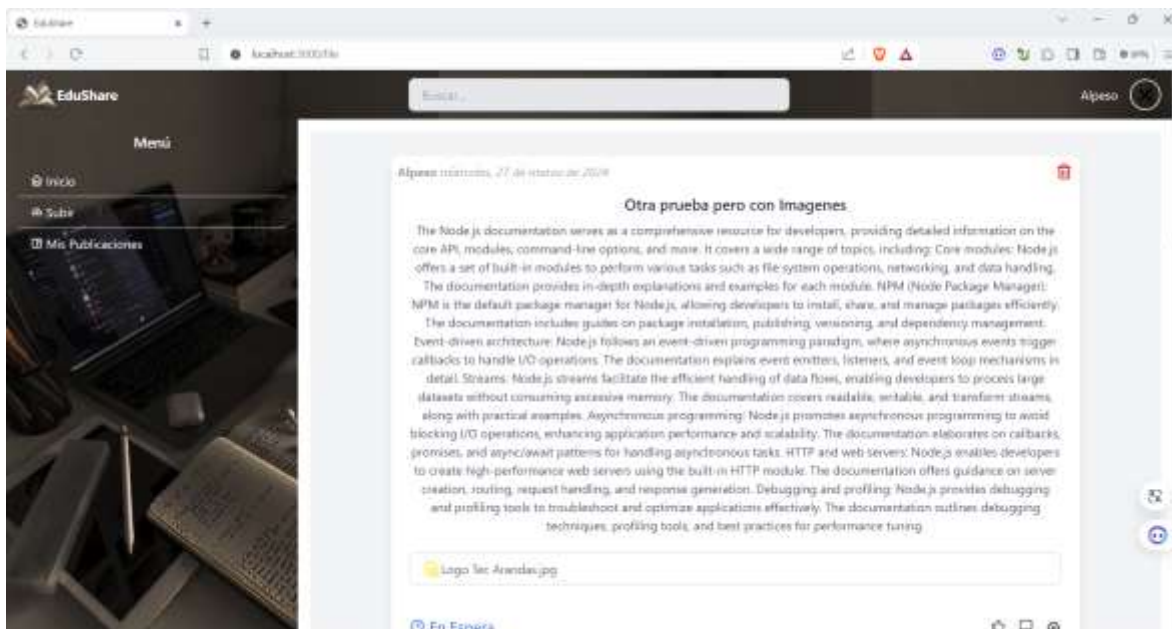
// Call the updatePets method
person.updatePets();

```

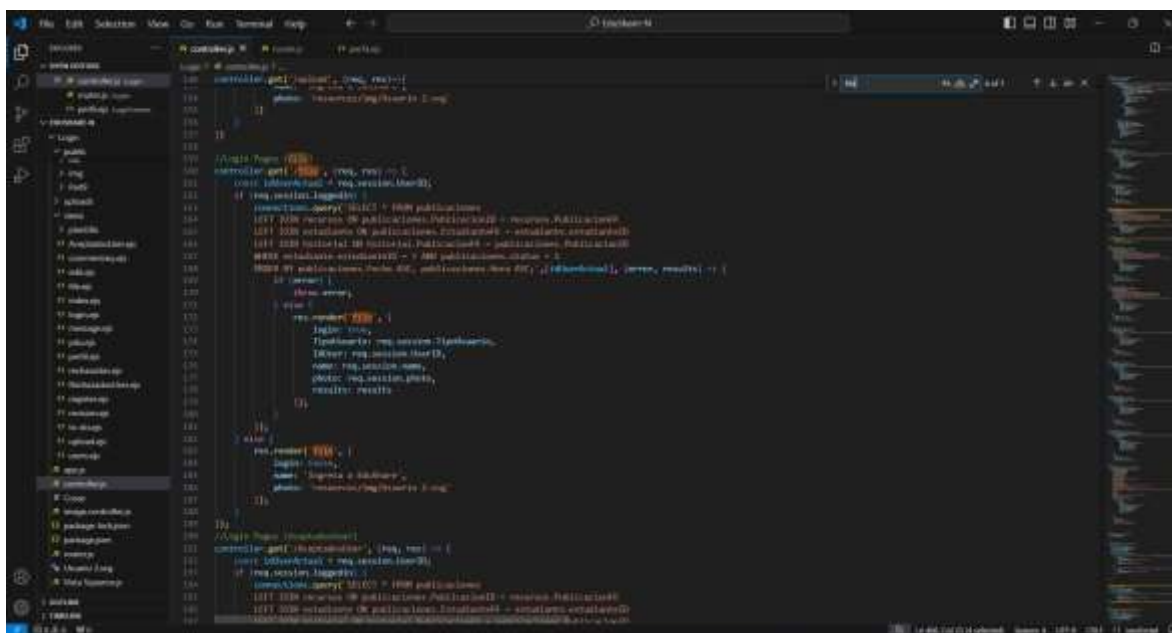
[illegible]

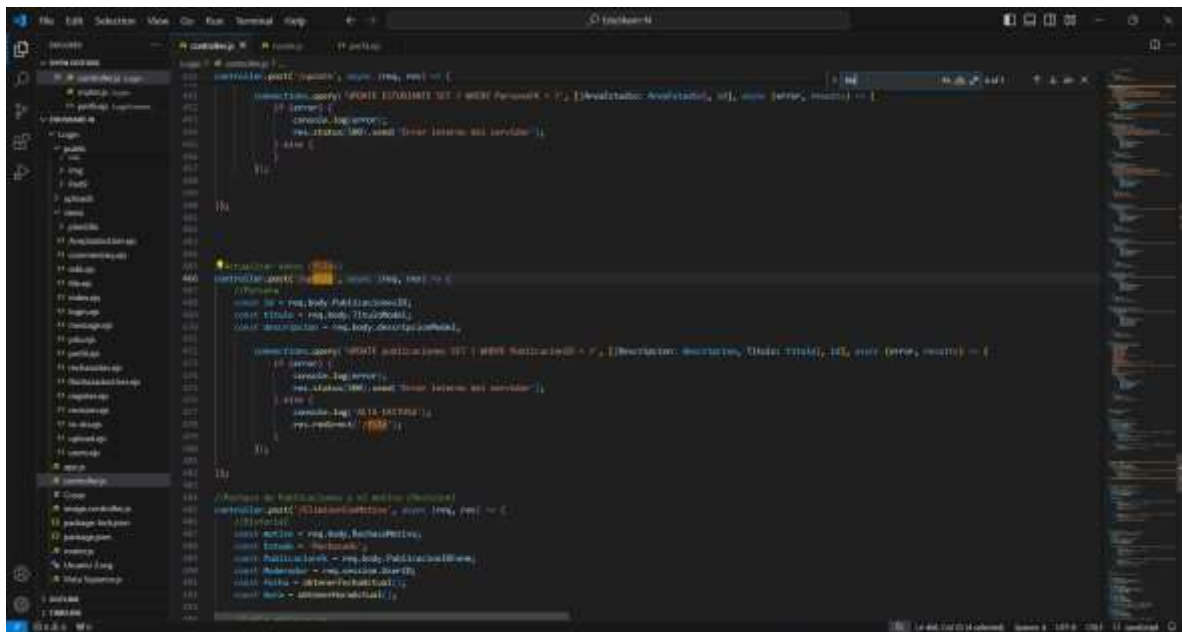
Mis Publicaciones

Este CRUD esta completo y es para la administración de las publicaciones, tiene tres apartados que son En espera: Son aquellas publicaciones recientes que los administradores todavía los tienen en lista de espera, y no ha habido alguna decisión



Código: esta es la consulta de vista y los Updates correspondientes a la eliminación lógica y al edit





```
const express = require('express');
const app = express();
const sequelize = require('./config/database');
const { Publicacion } = require('./models');

app.use(express.json());

// Crear una publicación
router.post('/publicaciones', async (req, res) => {
  const { titulo, contenido, fecha } = req.body;
  if (!titulo || !contenido || !fecha) {
    res.status(400).send('Error: faltan datos');
    return;
  }
  const nuevaPublicacion = new Publicacion({
    titulo,
    contenido,
    fecha,
  });
  await nuevaPublicacion.save();
  res.status(201).send('Publicación creada exitosamente');
});

// Actualizar una publicación
router.put('/:id', async (req, res) => {
  const { id } = req.params;
  const { titulo, contenido, fecha } = req.body;
  if (!titulo || !contenido || !fecha) {
    res.status(400).send('Error: faltan datos');
    return;
  }
  const publicacion = await Publicacion.findOne({ id });
  if (!publicacion) {
    res.status(404).send('Publicación no encontrada');
    return;
  }
  publicacion.titulo = titulo;
  publicacion.contenido = contenido;
  publicacion.fecha = fecha;
  await publicacion.save();
  res.status(200).send('Publicación actualizada exitosamente');
});

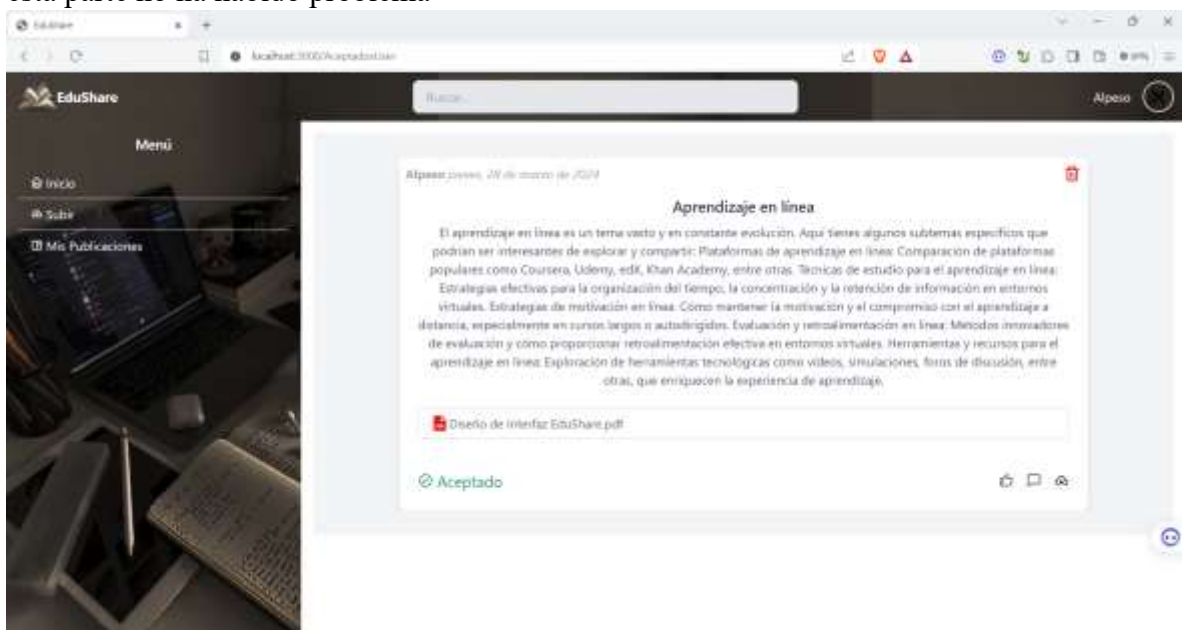
// Eliminar una publicación
router.delete('/:id', async (req, res) => {
  const { id } = req.params;
  const publicacion = await Publicacion.findOne({ id });
  if (!publicacion) {
    res.status(404).send('Publicación no encontrada');
    return;
  }
  await publicacion.destroy();
  res.status(200).send('Publicación eliminada exitosamente');
});
```



```
// Elimina y redirige a /file
router.delete('/:id', (req, res) => {
  const id = req.params.PublicacionID;
  connections.query('UPDATE publicaciones SET status = 4 WHERE PublicacionID = ?', [id], (error, results) => {
    if (error) {
      throw error;
    } else {
      res.redirect('/file');
    }
  });
});
```

Aceptados

En esta parte están los que el moderador ya dio un veredicto y están funcionando bien, en esta parte no ha habido problema



Código: esta es la consulta de vista y los Updates correspondientes a la eliminación lógica y al edit

```

// Elimina y redirige a /file
router.get('/defileRechazados/:PublicacionID', (req, res)=>{
  const id = req.params.PublicacionID;
  connections.query('UPDATE publicaciones SET status = 4 WHERE PublicacionID = ?', [id], (error, results)=>{
    if(error){
      throw error;
    } else {
      res.redirect('/AceptadosUser');
    }
  })
})

```

```

// Elimina y redirige a /file
router.get('/defileRechazados/:PublicacionID', (req, res)=>{
  const id = req.params.PublicacionID;
  connections.query('UPDATE publicaciones SET status = 4 WHERE PublicacionID = ?', [id], (error, results)=>{
    if(error){
      throw error;
    } else {
      res.redirect('/AceptadosUser');
    }
  })
})

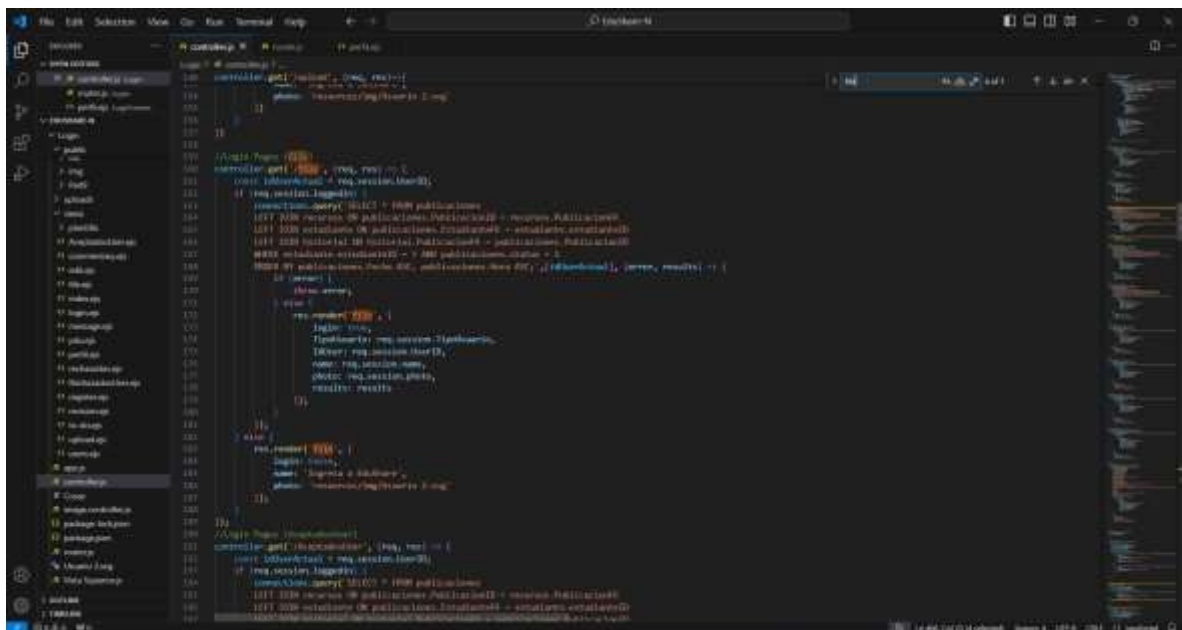
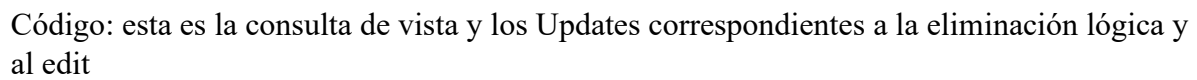
```

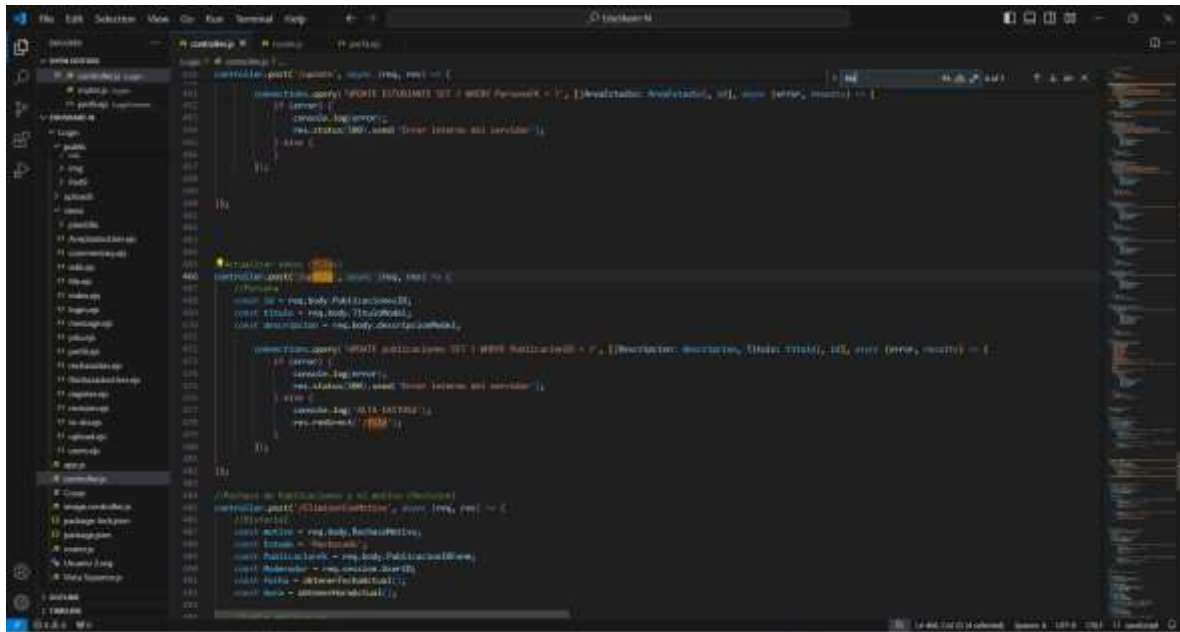
```

// Elimina y redirige a /file
router.get('/defileRechazados/:PublicacionID', (req, res)=>{
  const id = req.params.PublicacionID;
  connections.query('UPDATE publicaciones SET status = 4 WHERE PublicacionID = ?', [id], (error, results)=>{
    if(error){
      throw error;
    } else {
      res.redirect('/AceptadosUser');
    }
  })
})

```

En esta parte se muestran lo que no fueron aceptados y el motivo del porque no fueron aceptados





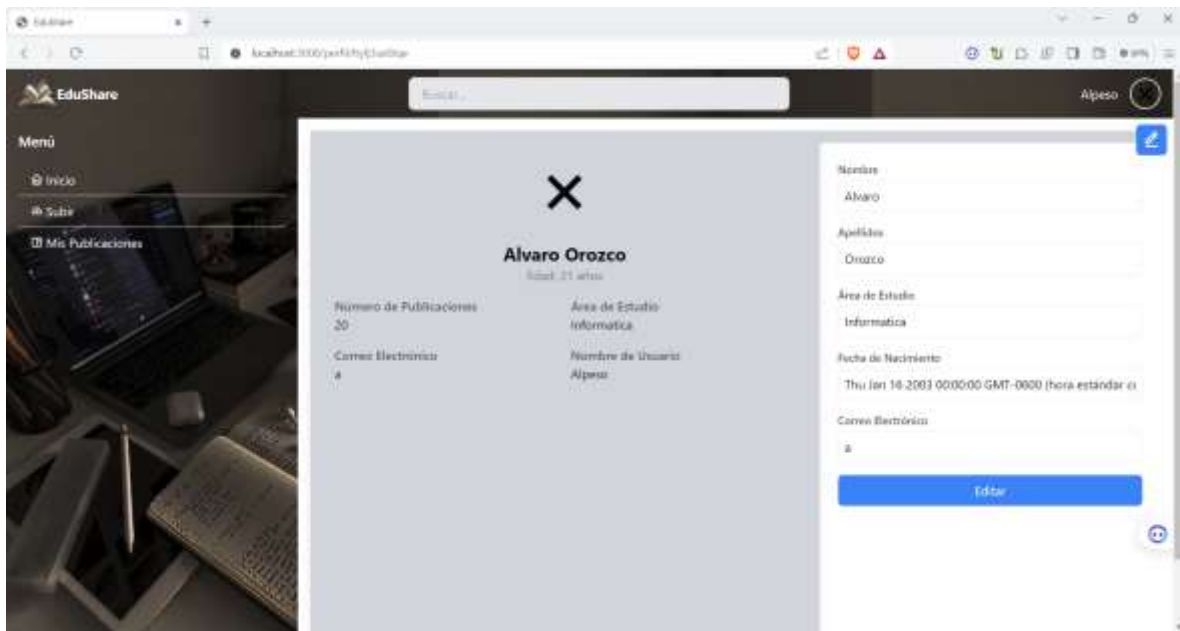
```

//Elimina y redirige a /file
router.get('/deleteRechazados/:PublicacionID', (req, res)=>{
  const id = req.params.PublicacionID;
  connections.query('UPDATE publicaciones SET status = 4 WHERE PublicacionID = ?', [id], (error, results)=>{
    if(error){
      throw error;
    } else {
      res.redirect('/AceptadosUser');
    }
  })
})

```

Perfil

Esto es para ver la información de los usuarios, y tomar una mejor decisión cuando se valla o no, aceptar una publicación de ese usuario



Código: Esta es solo la consulta en la base de datos que ayuda a que todo funcione correctamente

