Link de la app en producción
https://webserver-stack-angular.herokuapp.com/

Correo electrónico:
ing.osmarvera@gmail.com

Notes App

- **Backend**
  Se desarrollo 2 endpoint “/api/users” “/api/notes” en el cual se manejan el CRUD de los usuarios y las notas

```
class Server {

    constructor() {
        this.app = express();
        this.port = process.env.PORT;
        this.usersPath = '/api/users';
        this.notesPath = '/api/notes';

        // Conectar a base de datos
        this.conectarDB();

        // Middlewares
        this.middlewares();

        // Rutas de mi aplicación
        this.routes();
    }

    middlewares() {

        // CORS
        this.app.use(cors());

        // Lectura y parseo del body
        this.app.use(express.json());

        // Directorio Público
        this.app.use(express.static('public'));

    }

    async conectarDB() {
        await dbConnection();
    }
```

  En el cual se maneja la **conexión** a la base de datos **middlewares** para manejar el cors y la lectura de los request en json, también realizar estático el directorio público en el cual se pondrá el frontend

```
const mongoose = require('mongoose');

const dbConnection = async () => {
    try {

        await mongoose.connect(process.env.MONGO_URI, {
            useNewUrlParser: true,
            useUnifiedTopology: true,
            useCreateIndex: true,
            useFindAndModify: false
        });

    } catch (error) {
        console.log('-------------------------------------', error);
        throw new Error('Error al momento de inicializar la BD');
    }
}

module.exports = {
    dbConnection
};
```

Conexión a la base de datos de mongoDB

```
const { Router } = require('express');
const { check } = require('express-validator');
const { getNotes, getNote, createNote, updateNote, deleteNote } = require('../controllers/notes');
const { existeNotePorId, existeUsuarioPorId, validarFecha } = require('../helpers/db-validators');
const { validarCampos } = require('../middleware/validators');

const router = Router();
router.get('/', getNotes);

router.get('/:id', [
    check('id', 'No es un ID válido').isMongoId(),
    check('id').custom(existeNotePorId),
    validarCampos
], getNote);

router.post('/', [
    check('title', 'El titulo es obligatorio').not().isEmpty(),
    check('content', 'El contenido es obligatorio').not().isEmpty(),
    check('fecha', 'La fecha es obligatoria').not().isEmpty(),
    check('idUser', 'El usuario es obligatorio').not().isEmpty(),
    check('idUser', 'No es un ID válido').isMongoId(),
    check('idUser').custom(existeUsuarioPorId),
    check('fecha').custom(validarFecha),
    validarCampos
], createNote);

router.put('/:id', [
    check('id', 'No es un ID válido').isMongoId(),
    check('id').custom(existeNotePorId),
    validarCampos
], updateNote);

router.delete('/:id', [
    check('id', 'No es un ID válido').isMongoId(),
    check('id').custom(existeNotePorId),
    validarCampos
], deleteNote);
```

Archivo para manejar las peticiones al endpoint de Notes, en el cual se implementan validaciones de la paqueteria **express-validator** para realizar las principales validaciones y en caso de cumplir todas las validaciones acceda al controlador

```js
const Note = require('../models/Notes');

const { request, response } = require('express');

const getNotes = async (req = request, resp = response) => {
    const query = { activo: true };
    const notes = await Note.find(query).exec();
    return resp.status(200).json({ notes });
};

const getNote = async (req = request, resp = response) => {
    const { id } = req.params;
    const note = await Note.findById(id).exec();
    return resp.status(200).json({ note });
};

const createNote = async (req = request, resp = response) => {
    const { title, content, fecha, idUser } = req.body;
    const note = new Note({ title, content, fecha, idUser });

    await note.save();
    return resp.status(201).json({ note });
};

const updateNote = async (req = request, resp = response) => {
    const { id } = req.params;
    const note = await Note.findByIdAndUpdate(id, req.body).exec();
    return resp.status(200).json({ note });
};

const deleteNote = async (req = request, resp = response) => {
    const { id } = req.params;
    const note = await Note.findByIdAndDelete(id).exec();
    return resp.status(200).json({ note });
};
```

Archivo controlador el cual contiene la lógica de procesamiento de la información llegada, que son devolver todas las notas o una nota específica, también la creación de una nueva nota, actualizarla o eliminarla

```js
const { Schema, model } = require('mongoose');


const NotesSchema = Schema({
    title: {
        type: String
    },
    content: {
        type: String
    },
    fecha: {
        type: Date
    },
    activo: {
        type: Boolean,
        default: true
    },
    idUser: {
        type: Schema.Types.ObjectId,
        ref: 'User'
    },
});
```

Modelo de la nota en base de datos

```
const { Router } = require('express');
const { check } = require('express-validator');
const { validarCampos } = require('../middleware/validators');
const { getUsers, getUser, createUser, updateUser, deleteUser } = require('../controllers/users');
const { existeUsuarioPorId, userExiste } = require('../helpers/db-validators');

const router = Router();

router.get('/', getUsers);

router.get('/:id', [
    check('id', 'No es un ID válido').isMongoId(),
    check('id').custom(existeUsuarioPorId),
    validarCampos
], getUser);

router.post('/', [
    check('nombre', 'El nombre es obligatorio').not().isEmpty(),
    check('nombre').custom(userExiste),
    validarCampos
], createUser);

router.put('/:id', [
    check('id', 'No es un ID válido').isMongoId(),
    check('id').custom(existeUsuarioPorId),
    validarCampos
], updateUser);

router.delete('/:id', [
    check('id', 'No es un ID válido').isMongoId(),
    check('id').custom(existeUsuarioPorId),
    validarCampos
], deleteUser);
```

Archivo para manejar las peticiones al endpoint de users, en el cual se implementan validaciones de la paqueteria **express-validator** para realizar las principales validaciones y en caso de cumplir todas las validaciones acceda al controlador

```
const User = require('../models/Users');
const { request, response } = require('express');

const getUsers = async (req = request, resp = response) => {
    const query = { activo: true };
    const usuarios = await User.find(query);
    return resp.status(200).json({ usuarios });
};

const getUser = async (req = request, resp = response) => {
    const { id } = req.params;
    const usuario = await User.findById(id).exec();
    return resp.status(200).json({ usuario });
};

const createUser = async (req = request, resp = response) => {
    const { nombre } = req.body;
    const user = new User({ nombre });

    await user.save();
    return resp.status(201).json({ user });
};

const updateUser = async (req = request, resp = response) => {
    const { id } = req.params;
    const usuario = await User.findByIdAndUpdate(id, req.body).exec();
    return resp.status(200).json({ usuario });
};

const deleteUser = async (req = request, resp = response) => {
    const { id } = req.params;
    const usuario = await User.findByIdAndDelete(id).exec();
    return resp.status(200).json({ usuario });
};
```

Archivo controlador el cual contiene la lógica de procesamiento de la información llegada, que son devolver todos los usuarios o un usuario específico, también la creación de un nuevo usuario, actualizarlo o eliminarlo

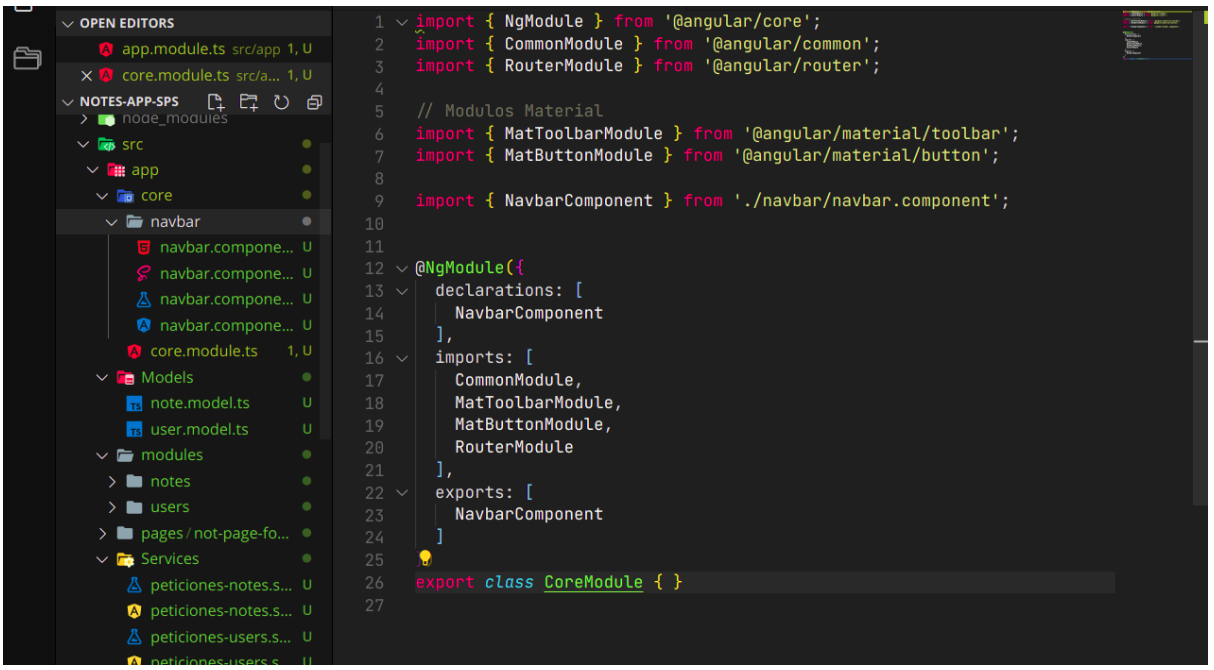
```
const { Schema, model } = require('mongoose');

const UserSchema = Schema({
    nombre: {
        type: String,
        required: [true, 'El nombre es obligatorio']
    },
    activo: {
        type: Boolean,
        default: true
    }
});
```

Modelo del usuario en la base de datos.

- **Frontend**
  Se desarrolló el frontend que consume las apis creadas en el backend mencionado anteriormente, en el cual para mejorar el diseño y la interacción del usuario se ocupa la librería de **material ui** y **sweetalert 2** para los mensajes con mejor diseño

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { ReactiveFormsModule } from '@angular/forms';
import { BrowserAnimationsModule } from '@angular/platform-browser/anim
import { RouterModule } from '@angular/router';
import { HttpClientModule } from '@angular/common/http';
import { FormsModule } from '@angular/forms';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

import { CoreModule } from './core/core.module';
import { NoteModule } from './modules/notes/note.module';
import { UserModule } from './modules/users/user.module';

import { NotPageFoundComponent } from './pages/not-page-found/not-page-

@NgModule({
  declarations: [
    AppComponent,
    NotPageFoundComponent,
  ],
  imports: [
    AppRoutingModule,
    BrowserAnimationsModule,
    BrowserModule,
    HttpClientModule,
    FormsModule,
    ReactiveFormsModule,
    RouterModule,
    CoreModule,
    NoteModule,
    UserModule,
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```

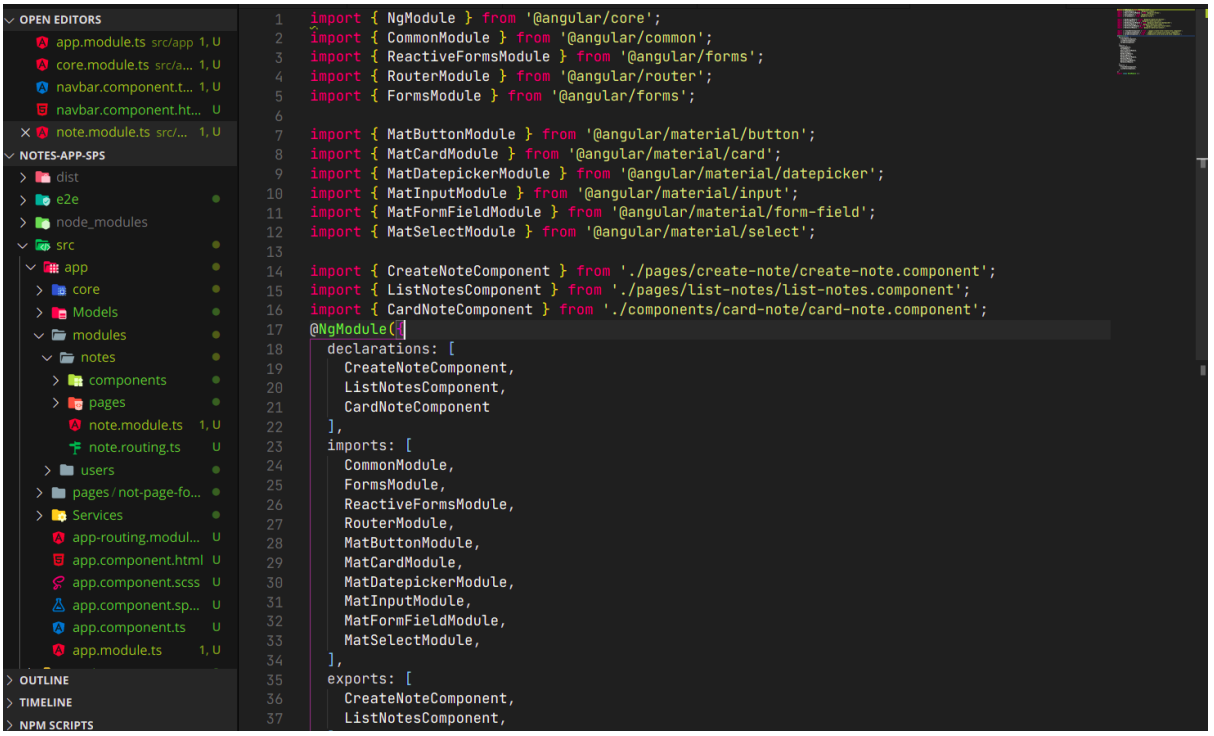
Se crearon 3 módulos para manejar de manera más ordenada los componentes que se ocuparon para las páginas creadas



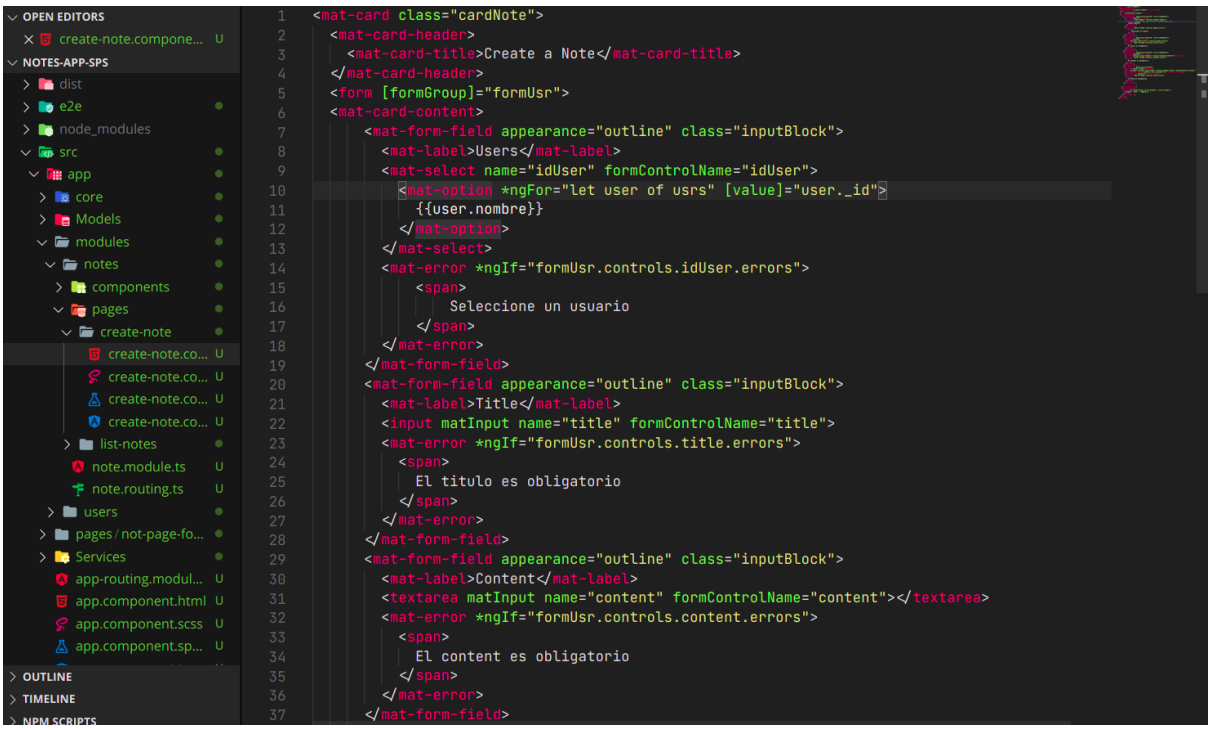
En el módulo de core se genera la **navbar** ya que esta será visible para todas las páginas, en el cual se importa **MatToolbarModule**, **MatButtonModule,** para el diseño de la navbar lo genere material ui

En la navbar se ocupa el componente toolbar de material y botones de referencia que dirige a las pantallas que indica el nombre al igual que un separador para poner a la izquierda el título y a la derecha las accione y el router-outlet muestre la pantalla indicada

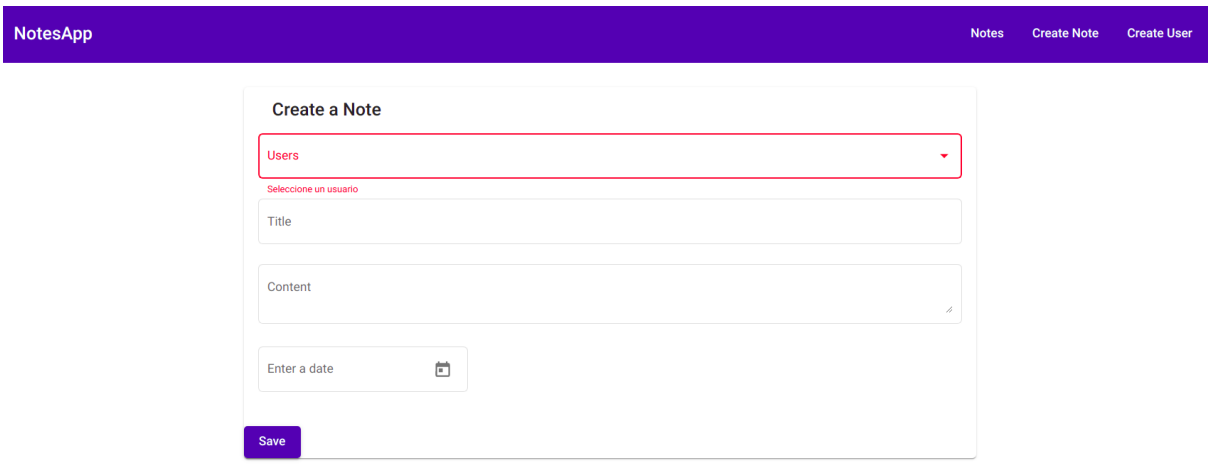
Muestra de resultado final del navbar

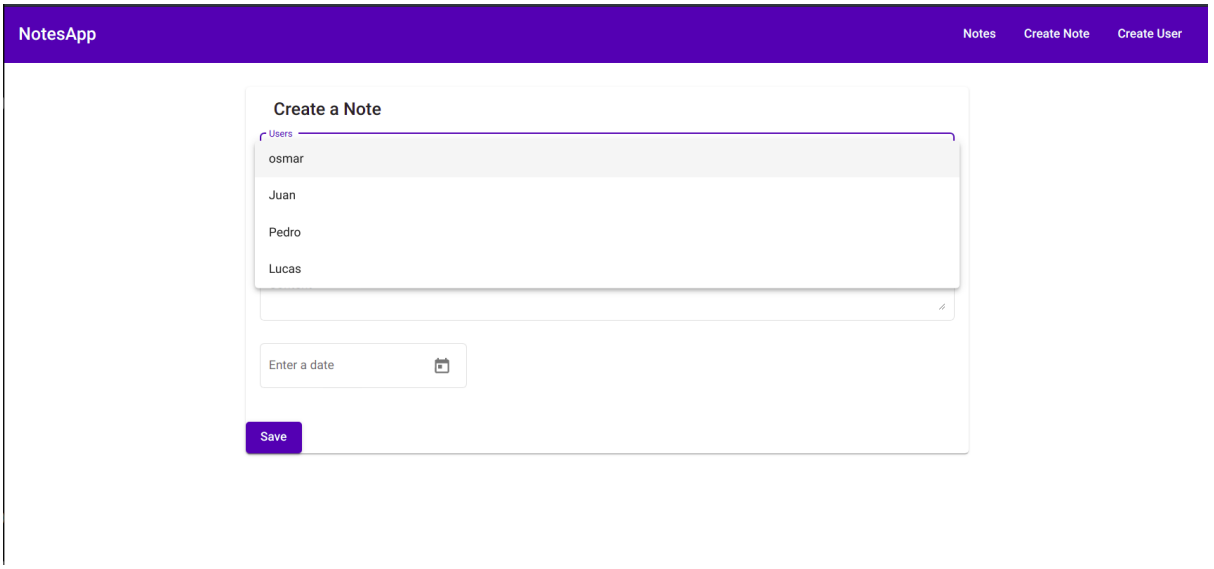


En el módulo de notes, se crean 4 componentes en el cual se crea y edita la nota que es **CreateNoteComponent** el componente **ListNotesComponent** en el cual se mostraran las cards de cada nota creada, en el componente **CardNoteComponent** es un componente separado en el cual se diseña 1 card específica



En el componente de crear nota se crea un formulario para la captura de la información necesaria para dar de alta la misma. En el componente select, se listan todos los usuarios creados.





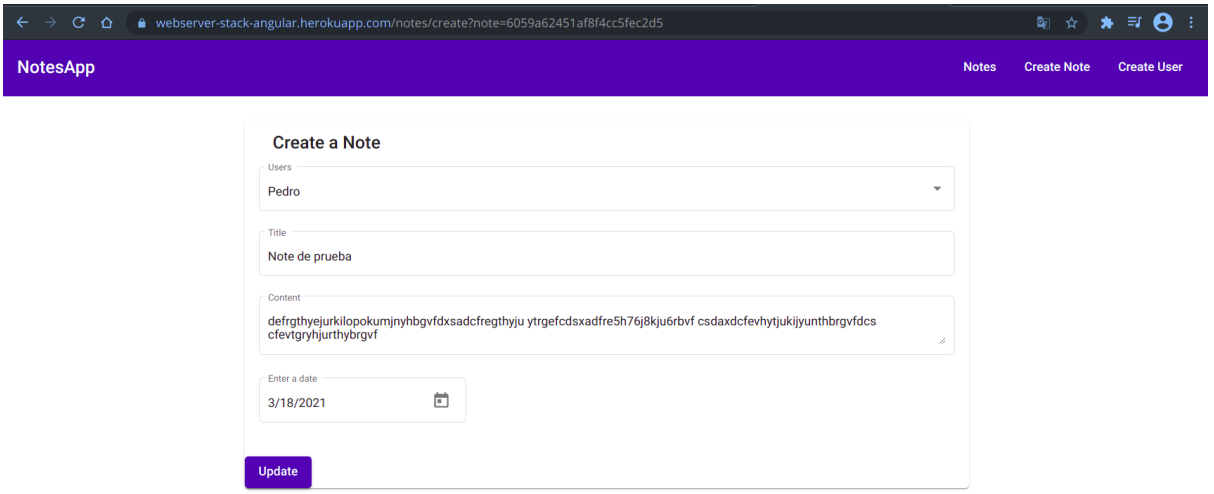
Muestra del resultado final del diseño de crear nota

```ts
export class CreateNoteComponent implements OnInit {
  public formUsr: FormGroup;
  public usrs: Array<any>;
  public edit: boolean;
  private idNota: string;

  constructor(
    private _route: ActivatedRoute,
    private createForms: FormBuilder,
    private noteSrv: PeticionesNotesService,
    private usrSrv: PeticionesUsersService,
  ) {
    this.edit = false;
    this.formUsr = this.createForms.group({
      idUser: ['', Validators.compose([Validators.required])],
      title: ['', Validators.compose([Validators.required])],
      content: ['', Validators.compose([Validators.required])],
      fecha: ['', Validators.compose([Validators.required])],
    });
    this.usrs = new Array();
  }

  ngOnInit(): void {
    this._route.queryParams.subscribe(({ note }) => {
      if (note) {
        this.edit = true;
        this.setNoteEdit(note);
      }
    });
    this.usrSrv.getUsers().subscribe((userss: any) => {
      this.usrs = userss;
    });
  }

```

Se ocupó **FormGroup** que mediante **FormBuilder** permite crear el formulario para validaciones de la información capturada, de igual forma se ocupa un **arreglo** en el cual se almacenarán los usuarios creados para ser mostrados en el componente select, de igual forma se crean 2 variables mas de tipo booleana y cadena para que se sepa si el usuario está editando una nota o creando una nueva, con **ActivatedRoute** se permite consultar los parámetros enviados por el url y si se tiene la note significa que se está editando.

```typescript
setNoteEdit(id: string) {
  this.idNota = id;
  this.noteSrv.getNote(id).subscribe(notaConsultada => {
    const { idUser, title, content, fecha } = notaConsultada;
    this.formUsr.patchValue({
      idUser, title, content, fecha
    });
  })
}

save() {
  this.edit ?
    this.updateNote()
    :
    this.saveNote();
}

saveNote() {
  if (!this.formUsr.valid) {
    Swal.fire('Error', 'Verifiquen todos los datos', 'error');
    return;
  }
  const { title } = this.formUsr.value;
  this.noteSrv.createNote(this.formUsr.value).subscribe(resp => {
    Swal.fire('Exito', `La nota ${title} fue registrada exitosamente`, 'success');
    this.formUsr.reset();
  });
}

updateNote() {
  if (!this.formUsr.valid) {
    Swal.fire('Error', 'Verifiquen todos los datos', 'error');
    return;
  }
  const { title } = this.formUsr.value;
  this.noteSrv.updateNote(this.formUsr.value, this.idNota).subscribe(resp => {
    Swal.fire('Exito', `La nota ${title} fue actualizada exitosamente`, 'success');
    this.formUsr.reset();
  });
}
```

En caso de estar editando se consulta la nota y se setea en el formulario creado



Resultado en caso de estar editando una nota

```typescript
export class PeticionesNotesService {

  constructor(
    private http: HttpClient
  ) { }

  createNote(formData: note) {
    return this.http.post(`${base_url}/notes`, formData).pipe(
      map((resp: any) => {
        return resp.msg;
      }),
      catchError(this.manejoError)
    );
  }

  deleteNote(id: string) {

    return this.http.delete(`${base_url}/notes/${id}`).pipe(
      map((resp: any) => {
        return resp.msg;
      }),
      catchError(this.manejoError)
    );

  }

  getNotes(): Observable<unknown> {

    return this.http.get(`${base_url}/notes`).pipe(
      map((resp: any) => {
        return resp.notes;
      }),
      catchError(this.manejoError)
    );
  }

```

Servicio que realiza la comunicación entre el backend y el frontend mediante el consumo de la api, realizando peticiones de post,get,put y delete, en caso de que el servidor responda un error se genera un error personalizado

```typescript
manejoError(error: HttpErrorResponse): Observable<never> {
  switch (error.status) {
    case 400:
      if (error.error.errors) {
        const errores = Object.keys(error.error.errors);
        Swal.fire('Error', error.error.errors[errores[0]].msg, 'error');
      } else {
        Swal.fire('Error', error.error.msg, 'error');
      }
      break;
    case 401:
      Swal.fire('Error', 'Esta accediendo sin autenticarse', 'error');
      break;
    case 404:
      const msg = Object.keys(error.error.errors);

      Swal.fire('Error', error.error.errors[msg[0]].msg, 'error');

      break;
    default:
      break;
  }
  return throwError('error inesperado');
}
```

Código que permite manejar los errores dados como respuesta por el servidor y permite mostrárselo al usuario mediante Swal para que este se entere que algo salio mal

```html
<div class="contenedor">
  <app-card-note [notes]="notas"></app-card-note>
</div>

```

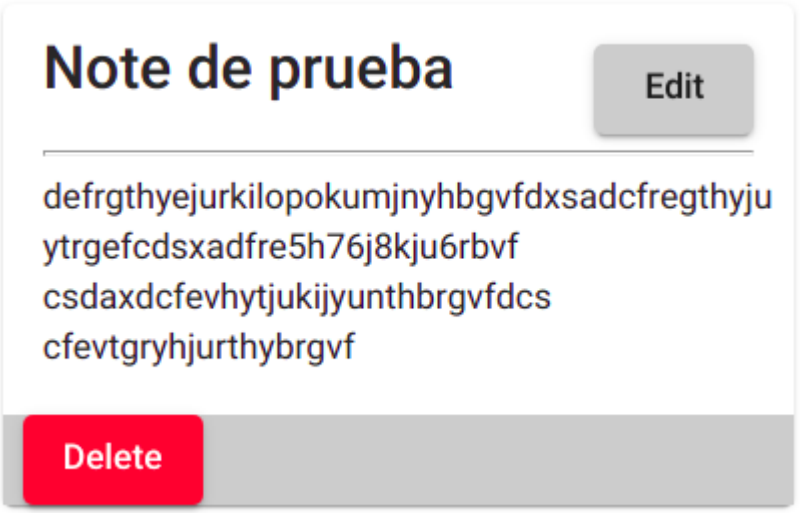
En el componente list-notes se ocupa el componente card-note como hija para dibujar las notas

```typescript
import { Component, OnInit } from '@angular/core';
import { PeticionesNotesService } from '../../../../Services/peticiones-notes.service';

@Component({
  selector: 'app-list-notes',
  templateUrl: './list-notes.component.html',
  styleUrls: ['./list-notes.component.scss']
})
export class ListNotesComponent implements OnInit {
  public notas: Array<any>;

  constructor(private notesSrv: PeticionesNotesService) {
    this.notas = new Array();
  }

  ngOnInit(): void {
    this.notesSrv.getNotes().subscribe((notesConsults: any) => {
      this.notas = notesConsults;
    });
  }

}
```

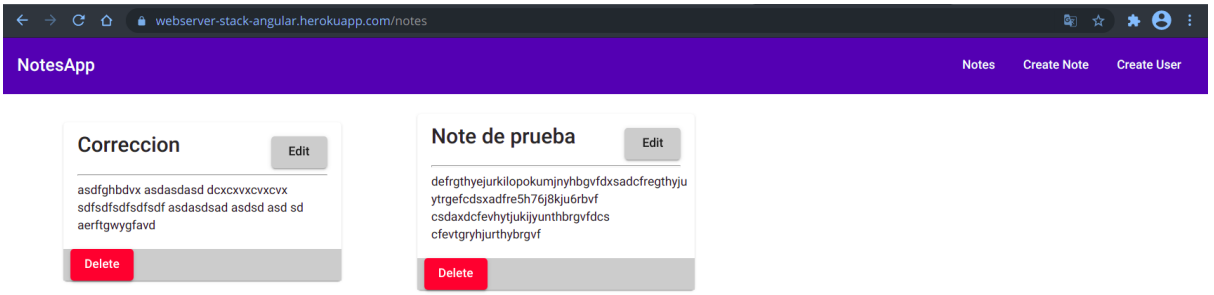
Controlador del componente list-notes realiza una peticion al servicio de peticionesService mostrada anteriormente en el cual le solicita al servidor que le mande todas las notes creadas y las almacena en la variable notas para enviarsela mediante **@Input** al componente hijo

```html
<div class="contenedorCardNotes">
  <mat-card class="cardNote" *ngFor="let note of notes; index as i">

    <div class="cardHeader">
      <mat-card-title>{{note.title}}</mat-card-title>

      <span class="separador"></span>

      <a mat-raised-button class="btnEdit" [routerLink]="['/notes/create']"
      [queryParams]="{note: note._id}"  routerLinkActive="router-link-active" >
        Edit
      </a>
    </div>

    <hr>
    <mat-card-content>

      {{note.content}}

    </mat-card-content>

    <mat-card-footer>
      <button mat-raised-button color="warn" class="btnDelete"
      (click)="deleteNote(note._id, i)">
        Delete
      </button>
    </mat-card-footer>
  </mat-card>
</div>
```

Codigo del componente hijo **card-note** en el cual se dibuja nota por nota



Resultado de una card dibujada



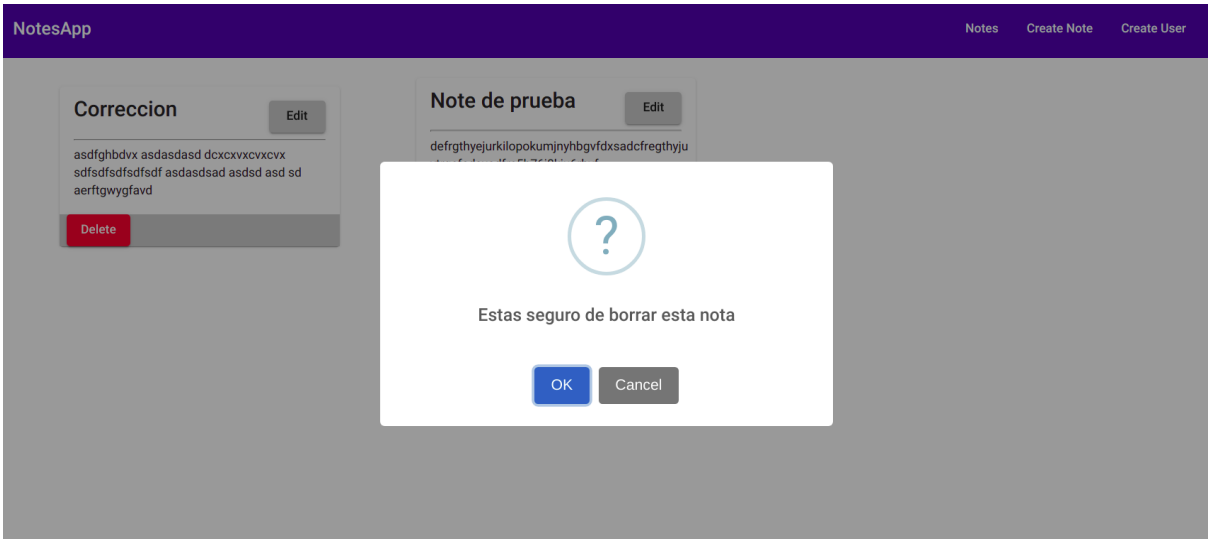
Dibujo de todas las cards dibujadas

```
<a mat-raised-button class="btnEdit" [routerLink]="['/notes/create']"
[queryParams]="{note: note._id}"  routerLinkActive="router-link-active" >
  Edit
</a>
```

Como se puede observar se tiene un botón que permite editar la nota el cual redirige a la pantalla antes mencionada mediante **queryParams** en el cual se envia el id de la nota

```
export class CardNoteComponent implements OnInit {
  @Input() notes: Array<any>;

  constructor(
    private noteSrv: PeticionesNotesService
  ) {
    this.notes = new Array();
  }

  ngOnInit(): void {
  }

  async deleteNote(id: string, pos: number) {

    const { value } = await Swal.fire({
      title: 'Estas seguro de borrar esta nota',
      icon: 'question',
      showCancelButton: true
    });
    if (value) {
      this.noteSrv.deleteNote(id).subscribe(resp => {
        this.notes.splice(pos, 1);
        Swal.fire('Exito', 'La nota fue eliminada correctamente', 'success');
      });
    }
  }
```

De igual forma se tiene un boton eliminar el cual ejecuta la función del controlador llamada **deleteNote** el cual recibe el id de la nota y su posición en el arreglo para que sea reactiva la eliminación de la nota, pero antes de esta se muestra un mensaje de confirmacion antes de eliminar la nota en caso de confirmarlo se borra



Resultado final del mensaje de confirmación antes de borrar la nota

```
<div class="contenedor">
  <mat-card class="cardUser">
    <mat-card-header>
      <mat-card-title>Create new user</mat-card-title>
    </mat-card-header>
    <form [formGroup]="formUsr">
      <mat-card-content>
        <mat-form-field appearance="outline">
          <mat-label>Nombre</mat-label>
          <input matInput name="nombre" formControlName="nombre">
          <mat-error *ngIf="formUsr.controls.nombre.errors">
            <span>
                Seleccione un usuario
            </span>
          </mat-error>
        </mat-form-field>
      </mat-card-content>
      <mat-card-footer>
        <button mat-raised-button color="primary" (click)="saveUsr()">
          Save
        </button>
      </mat-card-footer>
    </form>
  </mat-card>

  <span class="separador"></span>

  <mat-list class="listUsers">
    <div mat-subheader>Users</div>
      <mat-divider></mat-divider>
    <mat-list-item *ngFor="let user of usrs">
      <mat-icon mat-list-icon>person</mat-icon>
      <div mat-line>{{user.nombre}}</div>
      <mat-divider></mat-divider>
    </mat-list-item>
  </mat-list>
</div>
```

Código de la página de crear usuario en el cual se crea un formulario que solicita el nombre del usuario y un componente lista en el cual valga la redundancia se listan todos los usuarios creados

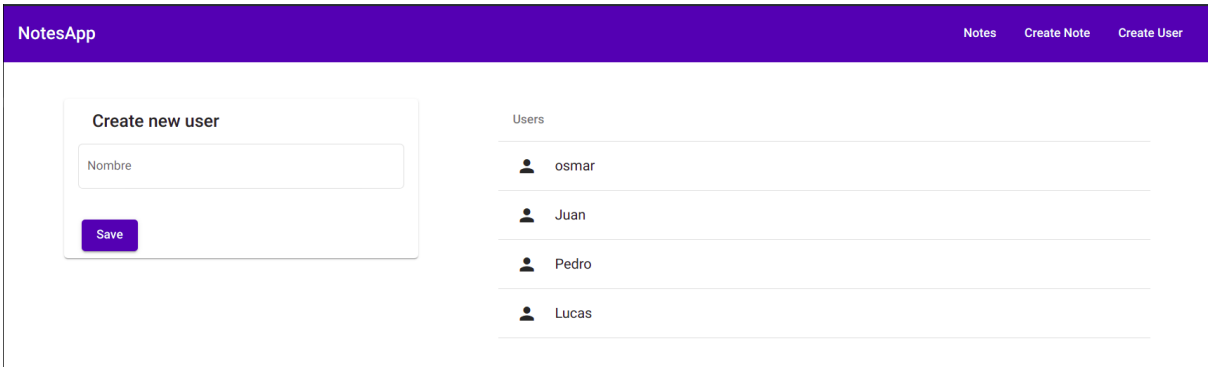
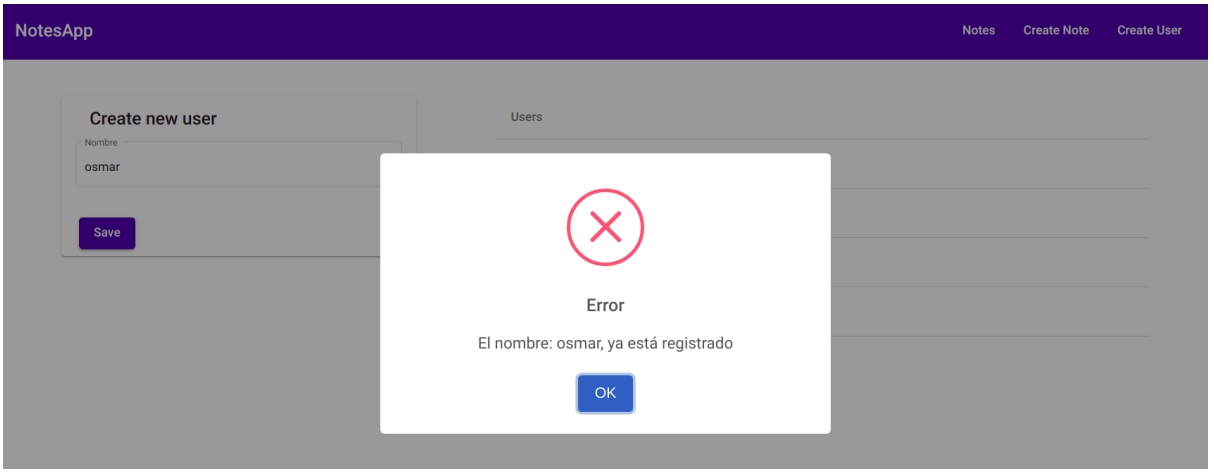


Imagen de la vista final de crear un usuario nuevo y la lista de los mismos

```
export class CreateUserComponent implements OnInit {
  public formUsr: FormGroup;
  public usrs: Array<any>;
  constructor(
    private createForms: FormBuilder,
    private usrSrv: PeticionesUsersService
  ) {
    this.formUsr = this.createForms.group({
      nombre: ['', Validators.compose([Validators.required])],
    });
    this.usrs = new Array();
  }

  ngOnInit(): void {
    this.usrSrv.getUsers().subscribe((userss: any) => {
      this.usrs = userss;
    });
  }

  saveUsr() {
    if (!this.formUsr.valid) {
      Swal.fire('Error', 'Captura el nombre', 'error');
      return;
    }

    const { nombre } = this.formUsr.value;

    this.usrSrv.createUser(this.formUsr.value).subscribe(resp => {
      Swal.fire('Exito', `El usuario ${nombre} fue registrado exitosamente`, 'success');
      this.usrs = [...this.usrs, resp];
      this.formUsr.reset();
    });
  }
```

Se ocupó **FormGroup** que mediante **FormBuilder** permite crear el formulario para validaciones de la información capturada, de igual forma se ocupa un arreglo de usuarios en el cual se almacenarán todos los usuarios registrados en la base de datos.

Como se puede observar el botón **Save** ejecuta la función **saveUsr** la cual realiza una validación si el formulario es válido, es decir capturaron la información obligatoria, en caso de cumplir la validación se registra el usuario. **Cabe recalcar** que en el backend se implementó una validación para que no se permita registrar 2 usuarios con el mismo nombre, no es tan estricta la validación con una combinación de mayúsculas o minúsculas pasa la validacion pero es una validación que se implemento para mostrar que se realizaron validaciones a nivel de registros en la base de datos.



Muestra el error en caso de duplicar los nombres de usuario.