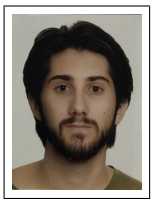
Notas - Una práctica de Angular



Asignatura: Programación Multimedia y Dispositivos Móviles
Curso: 2º CFGS Desarrollo de Aplicaciones Multiplataforma
Estudiante: Juan Miguel Rivas Velasco
Profesor: Carlos Serrano Sanchez
Fecha: 9 de noviembre de 2022

Índice

Índice de figuras

Resumen

Completar los puntos restantes de la práctica de Angular, para lograr una aplicación funcional que permita tomar notas. Así mismo, lograr desplegarla como PWA mediante el uso de Firebase.

- **Enlace al Github:** https://github.com/juanmirivas8/AngularPractica
- **Enlace a la aplicación desplegada:** https://practicaangular-15042.web.app

1. Introducción del proyecto

Este proyecto se trata de la primera toma de contacto con Angular, un framework para desarrollo Web centrado en componentes y servicios. Personalmente ha supuesto un desafío ya que auna diferentes tecnologías: TypeScript, HTML, CSS/SASS/SCSS, JSON, Firebase y FireStore, etc; además de aprender el propio funcionamiento del framework en sí.

Entre otras dificultades, lo más complejo ha sido aprender el sistema de componentes, servicios, inyección de dependencias ,routing con guards y por supuesto el sistema de importaciones y exportaciones con ngModule. Más sencillo sin embargo ha sido el aprender como se comunican los componentes entre ellos, usar elementos del ts en el DOM y viceversa.

Como es habitual me ha resultado tedioso aunque no especialmente complicado todo lo relacionado con la maquetación, diseño y uso de CSS.

Para finalizar con la introducción me hubiera gustado usar las promesas/callbacks directamente en un ejercicio ya que creo que es interesante, por ejemplo, implementar una animación de cargando mientras se cargan las notas, aunque finalmente no tuve tiempo de hacerlo, al igual que el modo oscuro.

2. Conceptos de Angular

2.1. Decorator

Los decoradores son aquellas instrucciones que empiezan por @ y que se colocan encima de una clase, método, propiedad o parámetro. Su función es inyectar código de manera que añaden muchisima funcionalidad de forma sencilla y rápida.

Algunos ejemplos de decoradores son:

- **Component**: Se utiliza para definir una clase como componente.
- **NgModule**: Se utiliza para definir una clase como módulo.
- **Injectable**: Se utiliza para definir una clase como servicio.
- **Directive**: Se utiliza para definir una clase como directiva.

2.2. ngModule

Ésta etiqueta se utiliza para definir un módulo que una unidad organizativa de componentes, directivas, servicios, pipes, etc. Para llevar a cabo esta tarea de coordinación, ésta etiqueta tiene, entre otros, los siguientes parámetros:

- **declarations**: indicamos las directivas,componentes, pipes, etc del módulo que queremos estén disponibles para usar por otros elementos del mismo módulo.
- **imports**: Indicamos que otros módulos queremos que estén disponibles para usar por otros elementos del módulo.
- **exports**: Indicamos qué componentes, directivas y pipes del módulo se exportan para que puedan ser utilizados por otros módulos.
- **providers**: Servicios y Guards que van a estar disponibles para todos los componentes del módulo como inyección de dependencias.
- **bootstrap**: Se indica sólo en el módulo que se va a cargar en primer lugar.

```
@NgModule({
  imports: [CommonModule, HighlightDirective, TranslateModule],
 declarations: [ NoteComponent ],
 exports:      [ NoteComponent/*, FormsModule */]
})
export class SharedModule { }
```

Figura 1: ejemplo ngModule

2.3. Component

Un componente es una clase que se utiliza para definir una vista, ya sea contenido de una página o elementos concretos como botones, navbars, etc. Ésta clase tiene un decorador @Component y se compone de tres partes principales:

- **selector**: Es el nombre del componente que se utiliza para insertar el componente en el DOM.
- **templateUrl**: Es la ruta del archivo HTML que contiene la vista del componente.
- **styleUrls**: Es la ruta del archivo CSS que contiene los estilos del componente.

Como nota comentar que tanto el HTML como el CSS pueden ser inline y que el componente puede crearse como standalone.

```
@Component({
  selector: 'app-form-note',
  standalone: true,
    imports: [CommonModule, ReactiveFormsModule, TranslateModule],
  templateUrl: './form-note.component.html',
  styleUrls: ['./form-note.component.css'],
})
```

Figura 2: Ejemplo componente

2.4. Service

Un servicio es una clase que se utiliza para definir una funcionalidad que se puede reutilizar en diferentes componentes. Los servicios se pueden inyectar en los componentes y se encargan de la lógica de la aplicación, es decir, las vistas o componentes sólo deben hacer tareas de interacción con el usuario y pasar los datos a los servicios.

La inyección se realiza en el constructor de la clase del componente pasado como argumento: constructor(private nombreServicio:Servicio), y debe haber sido declarado en el módulo del componente como provider.

2.5. Directives

2.5.1. Structural

Las directivas estructurales son aquellas que modifican la estructura del DOM, es decir, añaden o eliminan elementos del DOM.

Algunos ejemplos de directivas estructurales son:

- **<ngIf>:** Se utiliza para mostrar o no un elemento del DOM en función de una condición.
- **<ngFor>:** Se utiliza para iterar sobre un array y mostrar un elemento del DOM por cada elemento del array.
- **<ngSwitch>:** Se utiliza para mostrar un elemento del DOM en función de una condición.

Una forma elegante de usar ngIf es usando el contenedor ¡ng-container¿ya que no añade al DOM ningún elemento a diferencia de esconder un elemento.

```
<button [ngClass]="['languageButton']">
  <ng-container *ngIf="language === 'es'">
    <img src="assets/icons/united-kingdom-svgrepo-com.svg" (click)="changeLanguage( en: 'en')">
  </ng-container>
  <ng-container *ngIf="language === 'en'">
    <img src="assets/icons/spain-svgrepo-com.svg" (click)="changeLanguage( en: 'es')">
  </ng-container>
</button>
```

Figura 3: Ejemplo ngIf con ng-container

```
<div class="row m-1">
  <app-note [ngClass]="'col-sm-6 p-1'" *ngFor="let note of this.notesS.getNotes();let i=index;trackBy:trackByNotes" [note]="note"
            (editNote)="this.editingNote(note)" (removeNote)="this.removingNote(note)"></app-note>
</div>
```

Figura 4: Directiva Estructural

2.5.2. Attribute

Las directivas de atributo son aquellas que modifican el comportamiento de un elemento del DOM.

Algunos ejemplos de directivas de atributo son:

- **<ngClass>:** Se utiliza para añadir o eliminar clases CSS a un elemento del DOM.
- **<ngStyle>:** Se utiliza para añadir o eliminar estilos CSS a un elemento del DOM.
- **<ngModel>:** Se utiliza para enlazar un elemento del DOM con una variable del componente mediante two way data binding.

2.5.3. Custom

Podemos crear nuestras propias directivas y usarlas dentro de las etiquetas HTML añadiendo toda la funcionalidad programada de forma rápida.

2.6. Comunicación entre Componentes

2.6.1. Interpolation

La interpolación es la forma más sencilla de comunicación entre la vista y el TS. Mediante ella mostramos datos.

```
<h5 class="card-title">{{note.title}}</h5>
<p class="card-text">{{note.description}}</p>
```

Figura 5: Ejemplo Interpolation y pipes

2.6.2. Property Binding

Se trata de enlazar una propiedad de la etiqueta HTML con una variable del TS.

```
<button type="submit" class="btn btn-primary" [disabled]="this.form.status!='VALID'">
```

Figura 6: Ejemplo property binding

2.6.3. Event binding

Se trata de enlazar un evento de la etiqueta HTML, por ejemplo click, hover, drag, etc con una función del TS. A esta función podemos pasarle 0 o más parámetros e incluso el evento que se ha producido con su notación especial.

```
<app-form-note [note]="{id:0,description:'',title:''}" (onsubmit)="addNote($event)"></app-form-note>
```

Figura 7: Ejemplo event binding

2.6.4. Parent to child

Para pasar información de un componente padre a un componente hijo se utilizan los inputs y la asignacion de variables en el DOM.

```
export class NoteComponent implements OnInit {
   @Input('note') public note:INote = {
    id:-1,
    title:'',
    description:''
  };
```

Figura 8: Ejemplo parent to child - child

```
<app-form-note [note]="{id:0,description:'',title:''}" (onsubmit)="addNote($event)"></app-form-note>
```

Figura 9: Ejemplo parent to child - parent

2.6.5. Child to parent

Para comunicar un componente hijo con uno padre se usan los EventsEmitters con sus emits y outputs. El componente hijo emite un evento y el componente padre lo recibe y ejecuta una función.

```
@Output() editNote = new EventEmitter<INote>();
@Output() removeNote = new EventEmitter<INote>();
@ViewChild('boton') boton:any;
constructor() {
 }

ngOnInit(): void {
  console.log("NGONINIT")
}
ngAfterContentInit(){
  console.log("NGAFTERCONTENTINIT");
}

public editNoteFn(){
  if(this.note.id==-1) return;
  this.editNote.emit(this.note);
}
public removeNoteFn(){
  if(this.note.id==-1) return;
  this.removeNote.emit(this.note);
}
```

Figura 10: Ejemplo child to parent - child

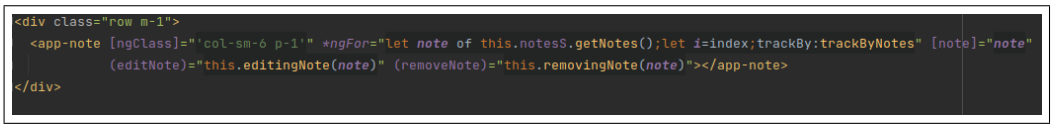


Figura 11: Ejemplo child to parent - parent

2.6.6. Reference Variable

Se trata de asignar un sobrenombre a un elemento del DOM para poder acceder a él desde el TS o desde otros elementos del DOM. Se utiliza el símbolo #nombre, y en el TS se accede a él con @ViewChild('nombre') nombre:ElementRef.

2.7. Pipes

Los pipes redirigen el contenido dentro del HTML y le aplican una transformación. Existen multitud de pipes implementados nativamente, pero también podemos crear nuestros propios pipes. Para ello mirar ejemplos en la documentación de Angular.

Figura 12: Ejemplo pipes y translate

2.8. Forms

Los formularios son una parte muy importante de cualquier aplicación web. Angular nos proporciona una serie de herramientas para crear formularios de forma sencilla y rápida. Para ello, Angular nos proporciona un módulo llamado FormsModule que debemos importar en el módulo principal de la aplicación. Podemos crear formularios manualmente con two-way binding y ngModel o bien podemos utilizar Formularios Reactivos con FormGroup y FormControl.

Figura 13: Ejemplo forms HTML

```
export class FormNoteComponent implements OnInit {

  @Input() note!:INote;
  @Output() onsubmit = new EventEmitter<INote>();
  public form:FormGroup;
  /*@ViewChild('title') title!:ElementRef;
  public description!:string;*/

  constructor(private fb:FormBuilder) {
    this.form = this.fb.group( controls: {
      title: ['',[Validators.required,Validators.minLength( minLength: 4)]],
      description:[''],
      id:['']
    })
  }
```

Figura 14: Ejemplo forms TS

2.9. Routing y Guards

El routing es la forma de navegar entre componentes. Para ello se utiliza el módulo de rutas de Angular. Además, podemos proteger las rutas con Guards, que son funciones que se ejecutan antes de entrar en una ruta y que pueden permitir o denegar el acceso a la misma y redirigir a otra ruta.

También podemos crear subrutas y cargar de forma lazy aquellos componentes que creamos muy pesados.

```
const routes: Routes = [
  {path:'a',component:AComponent,
  children:[
    {path:'suba', component:SubaComponent},
    {path:'subb',component:SubbComponent},
    {path:'**',redirectTo:'/a/suba', pathMatch:'full'}
  ]},
  {path:'b/:id/:page',component:BComponent},
  {path:'',redirectTo:'/a', pathMatch:'full'},
  {path:'**', component:Error404Component}
];
```

Figura 15: Ejemplo routing con subrutas

```
const routes: Routes = [
  {path:"home", component:NotesComponent ,
  canActivate:[LoginGuard]},
  {path:"new", component:NewComponent,
  canActivate:[LoginGuard]},
  {path:"about", loadComponent: ()=>import('./pages/about/about.component').then(c=>c.AboutComponent)},
  {path:'', redirectTo:'/home', pathMatch:'full'},
  {path:'login', component:LoginComponent,
    canActivate:[LoginGuard]},
  {path:'**', component:Error404Component}
];
```

Figura 16: Ejemplo guards

2.10. Firebase, PWA y deploy

Instalamos las dependencias de PWA con el comando: npm install @angular/pwa y hacemos el build con el comando: ng build Instalamos firebase-tools con npm install -g firebase-tools. Nos logueamos con firebase login y creamos un proyecto con firebase init. Finalmente desplegamos la aplicación con firebase deploy.

3. Objetivos de la práctica

3.1. Formateo de la lista

He optado por un sistema de cartas, usando las clases de Bootstrap.

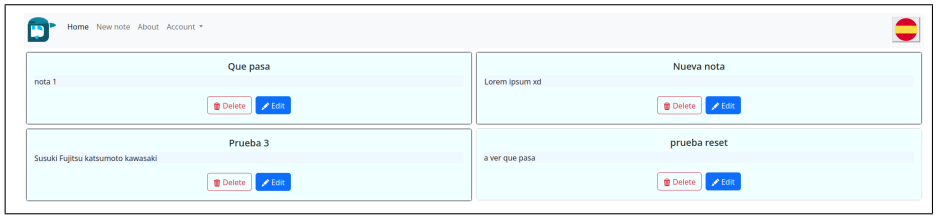


Figura 17: Formateo de la lista

3.2. Lista de tareas privada

Para lograr una lista de tareas privada simplemente he cambiado en el servicio de las notas el dbPath por /this.loginService.user.id tras inyectar el servicio de login en el constructor de notas.

```
export class NotesService {
  private dbPath = `/${this.loginService.user.id}`;
  notesRef!: AngularFirestoreCollection<any>;


  public notes:INote[] = [
  ];
  constructor(private db: AngularFirestore, private loginService:LoginService) {
    this.notesRef = db.collection(this.dbPath);
```

Figura 18: Servicio de notas

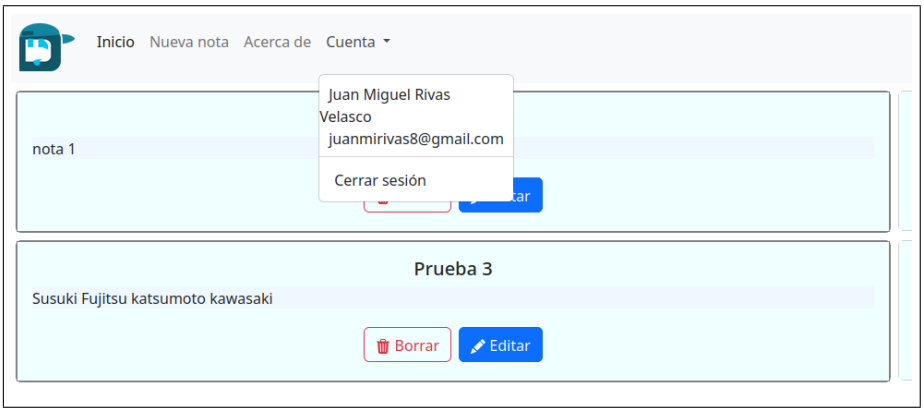


Figura 19: Lista de tareas privada - Cuenta 1

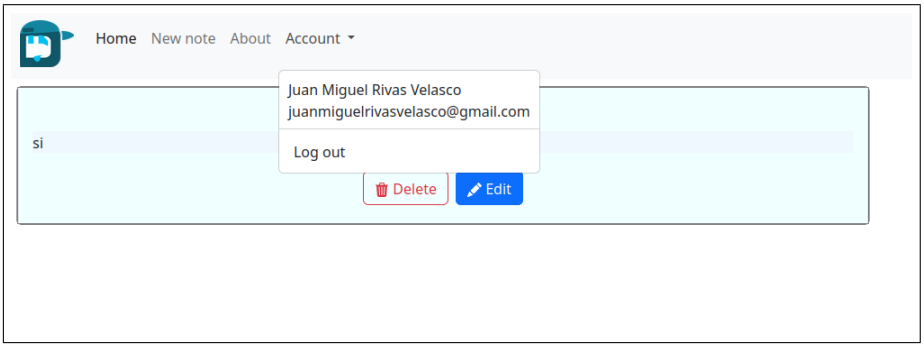


Figura 20: Lista de tareas privada - Cuenta 2

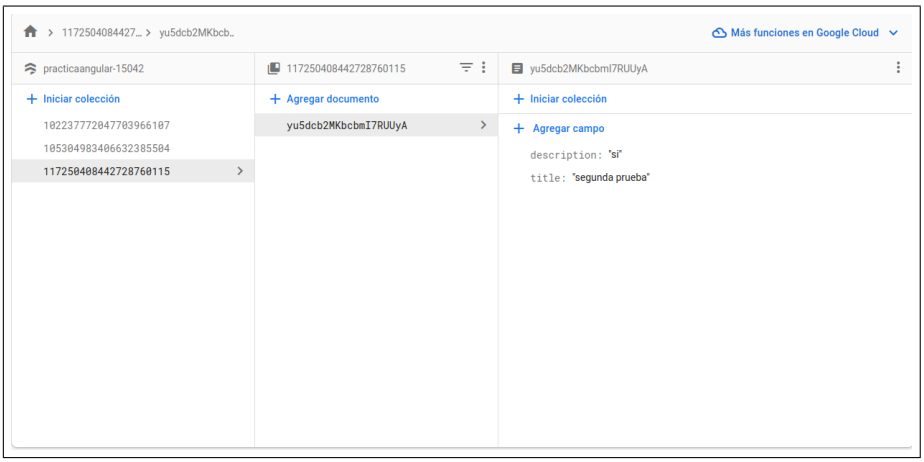


Figura 21: Lista de tareas privada - Firebase

3.3. Personalización de los iconos

He usado el icono que venía en el proyecto para el logo y he añadido dos svg para el botón de cambio de idioma.

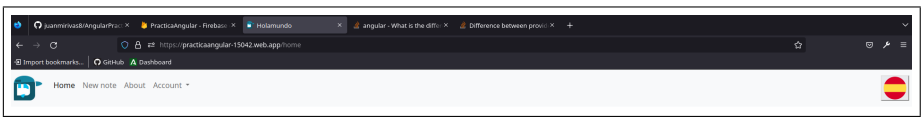


Figura 22: Personalización de los iconos

3.4. Página Acerca de

He intentado recrear un CV que tenía para practicar sobre bootstrap grid y flexbox.

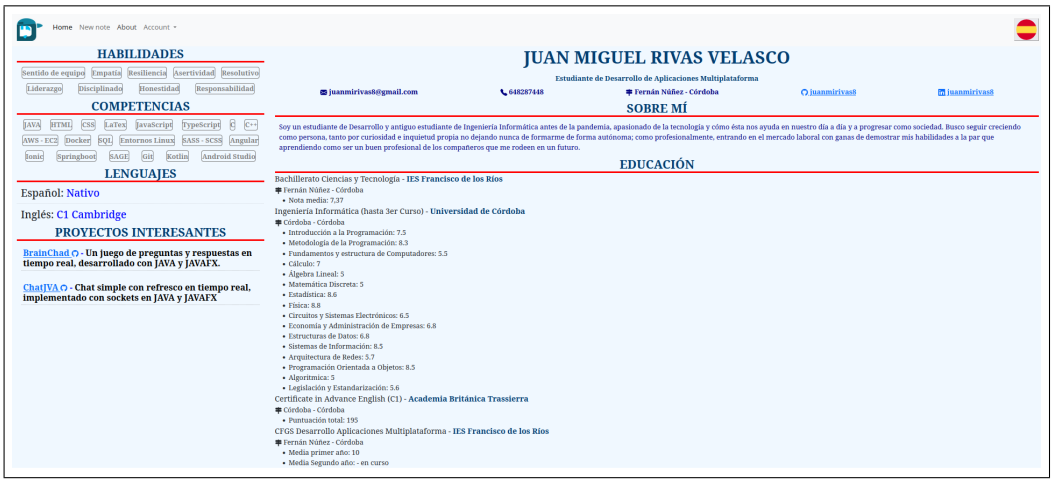


Figura 23: Página Acerca de

3.5. Login Persistente

Para lograr que el login sea persistente he usado el localStorage, que guarda los datos en el navegador del usuario.

```
constructor(private authService: SocialAuthService, private router:Router) {
    this.user = localStorage.getItem( key: 'user') ? JSON.parse(localStorage.getItem( key: 'user')!) : null;
    if(this.user){
      this.loggedIn=true;
      if(this.originalPath){
        this.router.navigate( commands: [this.originalPath]);
        this.originalPath='';
      }else
        this.router.navigate( commands: ['']);
    }else{
      this.authService.authState.subscribe( next: (user :SocialUser ) => {
        localStorage.setItem('user', JSON.stringify(user));
        this.user = user;
        this.loggedIn = (user != null);
        if(this.loggedIn){
          if(this.originalPath){
            this.router.navigate( commands: [this.originalPath]);
            this.originalPath='';
          }else
            this.router.navigate( commands: ['']);
        }else{
          this.router.navigate( commands: ['/login']);
        }
      });
    }
  }
```

Figura 24: Login persistente

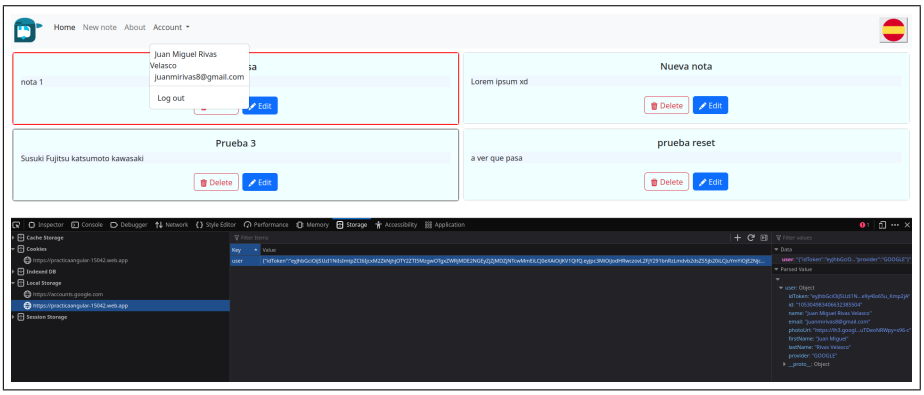


Figura 25: Login Persistente - Sesión Iniciada

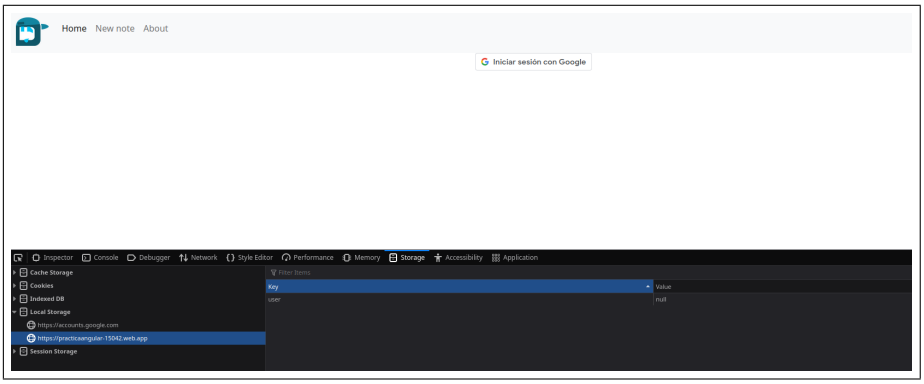


Figura 26: Login Persistente - Sesion Cerrada

3.6. Internacionalización

Para lograr la internacionalización he usado el paquete @ngx-translate/core y @ngx-translate/http-loader. Luego usamos archivos JSON para almacenar los pares de palabras y usamos el pipe translate en el HTML. Con un botón podemos cambiar el idioma mediante translate.use('es') o translate.use('en').

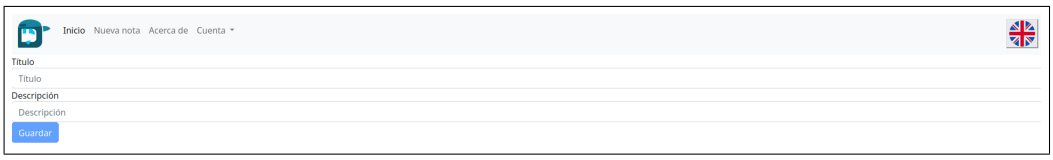


Figura 27: Internacionalización - Español

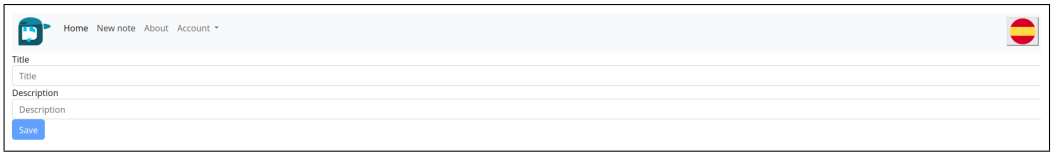


Figura 28: Internacionalización - Inglés