

Hack a note ~ API + BBDD + Front

◆ 1. Descripción

Debes crear una base de datos MySQL llamada notas, con una tabla llamada `lista_notas`, cuyos campos son: `id` como PK, NN y Autoincremental, y `texto`, que será un campo de texto largo (LONGTEXT).

◆ 2. Setup del proyecto

1. Crear el proyecto con `vue create hackanote`.
2. Instalar como dependencias Express y Nodemon: `npm i -D express nodemon`.
3. Instalar AXIOS `npm i --save axios`.
4. Instalar el conector MySQL `npm i --save mysql`.
5. Instalar bodyparser `npm i --save body-parser`.
6. Instalar cors `npm i --save cors`.

◆ 3. Configurando la API

Una vez instaladas todas las dependencias y librerías necesarias, crea dentro de la carpeta `src` un archivo llamado `api.js`.

Dentro, vamos a configurar todo lo necesario para que la API funcione y pueda conectarse con la base de datos.

Primero, vamos a importar las librerías necesarias:

```
const express = require('express')  
const cors = require('cors')  
const bodyParser = require('body-parser')  
const mysql = require('mysql')  
const app = express()
```

A continuación indicaremos a `app` (instancia de express) que utilice las dependencias:

```
app.use(cors())  
app.use(bodyParser.urlencoded({extended: true}))  
app.use(bodyParser.json())
```

Vamos ahora a crear la conexión con la base de datos que hemos creado. Te recuerdo que se llama `notas`.

```
// CONEXIÓN A LA BASE DE DATOS
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'password',
  database: 'notas'
})

// REALIZANDO CONEXIÓN CON LA BASE DE DATOS
connection.connect(error => {
  if(error) throw error
  console.log('DATABASE UP')
})
```

Finalmente, haremos que la API escuche el puerto `3050` en local:

```
// CONEXIÓN DE LA API
const PORT = 3050

app.listen(PORT, () => console.log('API UP'))
```

Asegúrate de que la base de datos está activa.

Finalmente vamos a crear 2 scripts adicionales dentro del archivo

`package.json`, uno será `main` y otro `start`:



```
{
  "name": "hackanote",
  "version": "0.1.0",
  "private": true,
  "main": "src/api.js", // 
  "scripts": {
    "serve": "vue-cli-service serve",
    "build": "vue-cli-service build",
    "start": "nodemon" // 
  }
}
```

Vamos a comprobar que las conexiones funcionan.

Abre 2 consolas del proyecto. En una, escribe `npm run serve`. En la otra, `npm start`.

Ambas conexiones deberían aparecerte sin problemas:

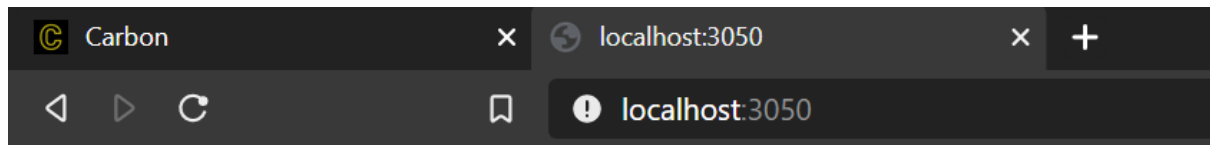
```
DONE Compiled successfully in 2500ms 23:24:54 (c) 2019 Microsoft Corporation. Todos los derechos reservados.
App running at:
- Local: http://localhost:8080/
- Network: http://192.168.1.140:8080/
Note that the development build is not optimized.
To create a production build, run npm run build.
C:\Users\dmcha\OneDrive\Escritorio\VUE_HAB\TESTS\hackanote\hackanote>npm start
> hackanote@0.1.0 start C:\Users\dmcha\OneDrive\Escritorio\VUE_HAB\TESTS\hackanote\h
ackanote
> nodemon
[nodemon] 2.0.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting 'node src/api.js'
API UP
DATABASE UP
```

Si te aparece algún problema, comprueba los pasos que has hecho.

A continuación crea una llamada GET simple en la API, una que solo mande una respuesta estática:

```
// ENVIANDO MENSAJE SIMPLE EN UN GET
app.get('/', (req, res) => {
  res.send('TE DOY LA BIENVENIDA A MI API 🙌')
})
```

Y accede a través del navegador a la url `localhost:3050`, donde tu API está trabajando. Deberías ver algo similar a esto:



TE DOY LA BIENVENIDA A MI API 🙌

Acabamos de comprobar que el servicio de la API funciona, es hora de crear las llamadas.

◆ 4. Creando las llamadas GET, POST, PUT y DELETE

Por ahora vamos a crear unas llamadas muy simples para setear las URLs de la API.

Al crearlas, vamos a ir llamándolas desde el postman para ver que realmente las rutas funcionan.

Recogiendo todas las notas

```
// RECOGIENDO TODAS LAS NOTAS
app.get('/notas', (req, res) => {
  res.send('Mis notas 📁')
})
```

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** localhost:3050/notas
- Buttons:** Send, Params, Authorization, Headers (6), Body, Pre-request Script, Tests, Settings.
- Query Params Table:**

KEY	VALUE	DESCRIPTION
Key	Value	Description
- Body Tab:** Status: 200 OK, Time: 29 ms, Size: 250 B, Save.
- Body Content:** Pretty, Raw, Preview, Visualize, HTML. The response body is: 1 Mis notas 📁.

Recogiendo una nota específica

```
// RECOGIENDO UNA NOTA
app.get('/notas/:id', (req, res) => {
  res.send('Mi nota: 📝')
})
```

GET localhost:3050/notas/3

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

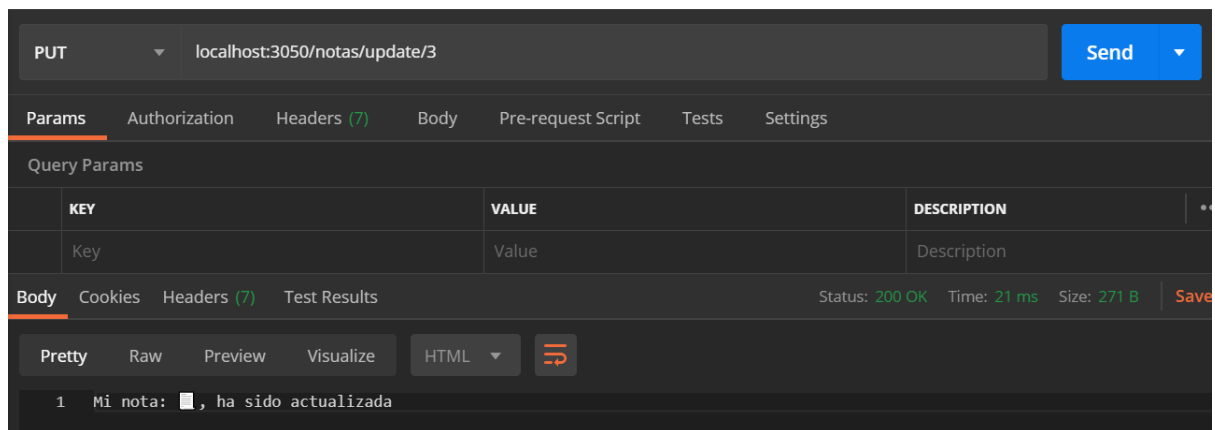
Body Cookies Headers (7) Test Results Status: 200 OK Time: 9 ms Size: 249 B Save

Pretty Raw Preview Visualize HTML

1 Mi nota: 📝

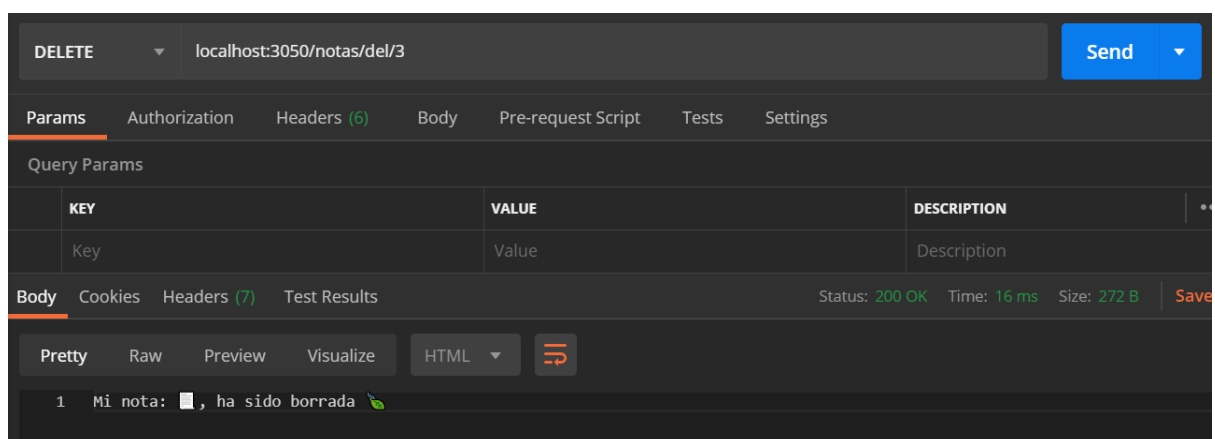
Actualizando una nota

```
// ACTUALIZANDO UNA NOTA
app.put('/notas/update/:id', (req, res) => {
  res.send('Mi nota: 📝, ha sido actualizada')
})
```

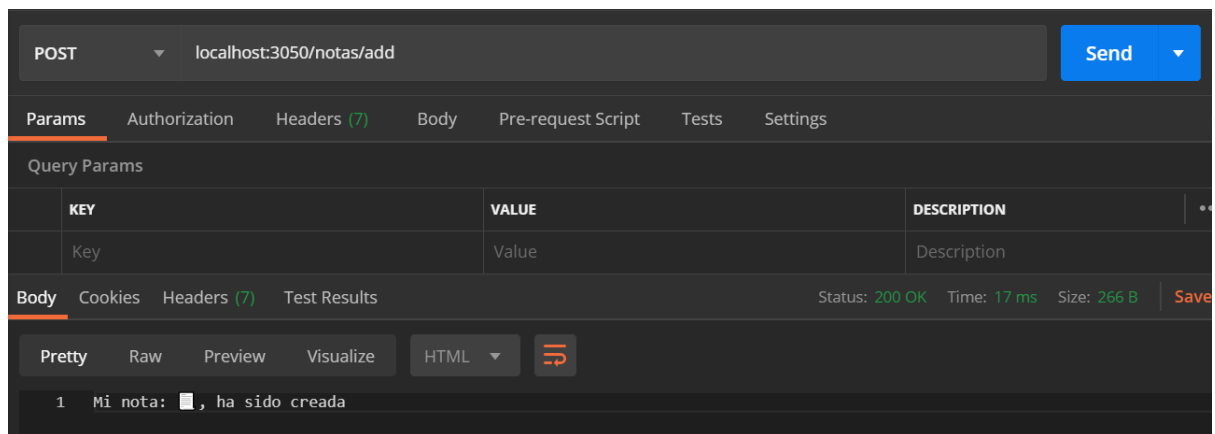
Borrando una nota

```
// BORRANDO UNA NOTA
app.delete('/notas/del/:id', (req, res) => {
  res.send('Mi nota: [icon], ha sido borrada [icon]')
})
```



Añadiendo una nota

```
// AÑADIENDO UNA NOTA
app.post('/notas/add', (req, res) => {
  res.send('Mi nota: 📝, ha sido creada')
})
```



Una vez hemos comprobado a través de Postman que todas las URLs son válidas, podemos empezar a crear la lógica.

◆ 5. Preparando el front

Para este proyecto vamos a crear las vistas: `About.vue` y `Notas.vue`.

En `Notas.vue` crearemos toda la lógica y mostraremos las notas, en `About.vue` deberás colocar algo de información sobre ti y sobre con qué has realizado este proyecto.

En `Notas.vue` por ahora vamos a implementar un simple formulario:

```
<h1>
  ¡Hola! 🙌
</h1>

<h3>
  Puedes crear tu nota a través del formulario aquí abajo 📌
</h3>

<div class="add">

  <form>

    <label for="textarea">Texto de la nota:</label>
    <br>

    <textarea v-model="texto" id="textarea"
              name="textarea" cols="40" rows="5">
    </textarea>
    <br>

    <button>
      CREAR
    </button>

  </form>

</div>
```

¡Hola! 🤝

Puedes crear tu nota a través del formulario aquí abajo 📍

Texto de la nota:

CREAR

Este formulario lo utilizaremos para hacer nuestras peticiones `post`.

Fíjate que en el `textarea` se le ha agregado un `v-model` de una variable llamada `texto`. Crea esa variable en el objeto `data` de la vista:

```
data(){  
  return {  
    texto: ''  
  }  
}
```

◆ 6. Creando la primera nota con POST

Para crear notas, debemos antes crear la lógica, tanto en la API como en el front.

Primero, nos dirigimos a `api.js` y vamos a rehacer la llamada `post` que teníamos, primero creando la secuencia sql que creará las notas:

```
// AÑADIENDO UNA NOTA
app.post('/notas/add', (req, res) => {
  // SECUENCIA SQL
  const sql = 'INSERT INTO lista_notas SET ?'
})
```

La base de datos espera un objeto. Si bien es cierto que sólo tenemos el campo `texto`, igualmente debemos pasarlo como objeto.

```
// AÑADIENDO UNA NOTA
app.post('/notas/add', (req, res) => {
  // SECUENCIA SQL
  const sql = 'INSERT INTO lista_notas SET ?'
  // OBJETO QUE RECIBE LA BASE DE DATOS
  const newNote = {
    texto: req.body.texto
  }
})
```

A continuación establecemos conexión con la base de datos, ejecutando en la llamada esa secuencia sql que hemos creado:

```
// AÑADIENDO UNA NOTA
app.post('/notas/add', (req, res) => {
  // SECUENCIA SQL
  const sql = 'INSERT INTO lista_notas SET ?'
  // OBJETO QUE RECIBE LA BASE DE DATOS
  const newNote = {
    texto: req.body.texto
  }
  // CONEXIÓN Y EJECUCIÓN DEL SQL
  connection.query(sql, newNote, error => {
    if(error) throw error
    res.send('Nota creada')
  })
})
```

La lógica de la función está hecha, ahora se ha de utilizar desde el front.

Volvemos a `Notas.vue`, donde vamos a importar AXIOS:

```
import axios from 'axios'
```

Y a continuación crearemos una función que ejecute la llamada de **AXIOS** con el método post y envíe a la base de datos nuestra petición:

```
addNote(){
  // CAMBIAMOS EL SCOPE DENTRO DE LA LLAMADA
  var self = this
  // REALIZAMOS UN POST A LA URL DE LA API
  axios.post('http://localhost:3050/notas/add', {
    // ENVIAMOS LA VARIABLE 'TEXTO', ENLAZADA
    // AL TEXTAREA DEL FORMULARIO
    texto : self.texto
  })
  // SI SALE BIEN
  .then(function (response) {
    console.log(response)
  })
  // SI SALE MAL
  .catch(function (error) {
    console.log(error)
  })
}
```

Finalmente, debemos enlazar esta función con el botón del formulario:

```
<button @click="addNote()">
  CREAR
</button>
```

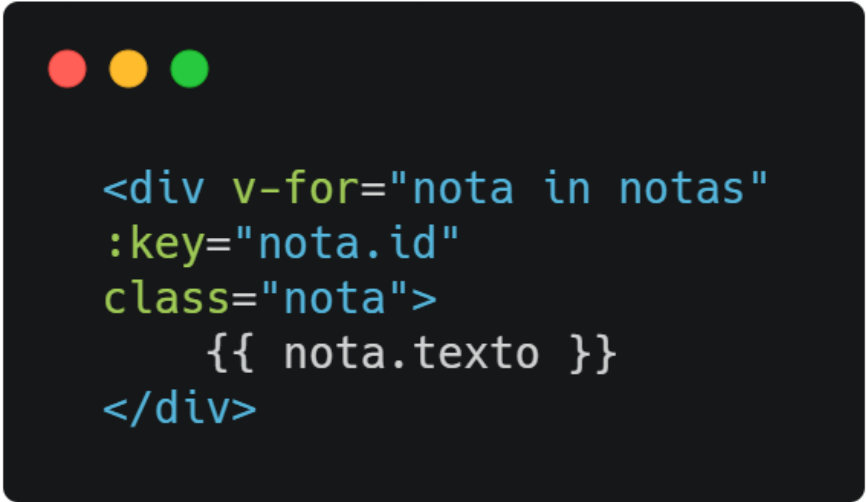
Prueba a poner algo de texto en el formulario y enviarlo, ¡ahora podrás crear tus notas!

◆ 7. Mostrando las notas por pantalla con GET

Es hora de mostrar por pantalla las notas que tenemos.

Primero, vamos a crear un componente llamado `ShowNotas.vue`, donde estilizaremos las notas.

Crearemos directamente un `v-for`, ya sabemos que las notas vienen con su único atributo `texto`, con lo cual:



```
<div v-for="nota in notas"
:key="nota.id"
class="nota">
  {{ nota.texto }}
</div>
```

Estilizamos un poco el componente:


```
.nota {
  margin: auto;
  margin-top: 0.667rem;
  padding: 0.400rem;
  width: 50%;
  border-radius: 15px;
  border: 1px solid slateblue;
  font-size: 1.2rem;
  box-shadow: 2px 4px 10px lightgray;
}
```

Le indicamos que va a recibir un array como prop:

```
export default {
  name: 'ShowNotas',
  props: {
    notas: Array
  }
}
```

Lo importamos, declaramos e insertamos en `Notas.vue`:

```
import axios from 'axios'
import notas from '@/components/ShowNotas.vue'

export default {
  name: 'Notas',
  components: {
    notas
  }
}
```

```
<h2>
  Notas
</h2>
<notas :notas="notas"></notas>
```

Pese a que todavía no existe ninguna variable en `Notas.vue` que se llame `notas` y que podamos enviar al componente, en un momento la estaremos creando.

Volvemos a `api.js`, donde debemos crear la lógica de la llamada para recoger todas las notas de la base de datos:

```
// RECOGIENDO TODAS LAS NOTAS
app.get('/notas', (req, res) => {
  // SECUENCIA SQL
  const sql = 'SELECT * FROM lista_notas'
  // CONEXIÓN Y EJECUCIÓN DEL SQL
  connection.query(sql, (error, results) => {
    // SI HAY ERROR, QUE LO MUESTRE
    if(error) throw error
    // COMPROBAR QUE LA RESPUESTA NO VIENE VACÍA
    if(results.length > 0) {
      res.json(results)
    }
    // EN CASO DE QUE VENGA VACÍA
    else {
      res.send('Lista de notas no encontrada')
    }
  })
})
```

Debemos volver ahora a `Notas.vue` para crear una nueva variable de tipo array llamada `notas` que recoja las notas, y para crear la llamada de AXIOS que ejecute la llamada a la base de datos, llamada `getNotes()`:

```
data(){  
  return {  
    texto: '',  
    notas: []  
  }  
}
```

```
getNotes(){  
  
  var self = this  
  // HACIENDO LA LLAMADA A LA API/BBDD  
  axios.get('http://localhost:3050/notas')  
  .then(function (response) {  
    // RELLENANDO EL ARRAY DE NOTAS  
    self.notas = response.data  
  })  
  .catch(function (error){  
    console.log(error)  
  })  
}
```

Llamaremos esta función en un hook `created` para que se ejecute cuando la página se cree:

```
// CREANDO EL HOOK CREATED
created(){
  // LLAMANDO A GETNOTES EN CUANTO LA
  // PÁGINA SE CREA
  this.getNotes()
}
```

Ahora, tus notas deberían aparecer en pantalla:

Notas 🖱️

Nota de prueba

Ir de compras

Ir a pescar

◆ 8. Actualizando las notas con PUT

En este caso vamos a actualizar las notas con PUT.

Creamos la lógica correspondiente en `api.js`:

```

app.put('/notas/update/:id', (req, res) => {
  const texto = req.body.texto
  const id = req.body.id
  const sql = `UPDATE lista_notas SET texto='${texto}' WHERE id=${id}`
  connection.query(sql, error => {
    if(error) throw error
    res.send('Nota actualizada')
  })
})

```

En `ShowNotas.vue` vamos a modificar un poco el HTML, añadir un index al v-for y añadir un botón que ponga "EDITAR", que al ser clicado hace trigger de un evento llamado `editNoteEvent`, el cual recibe el index de cada nota que se seleccione:

```

<div v-for="(nota, index) in notas"
  :key="nota.id"
  class="nota">
  {{ nota.texto }}
  <button @click="editNoteEvent(index)">
    EDITAR
  </button>
</div>

```

A continuación, en `methods` creamos dicho evento:



```
editNoteEvent(index){  
    let data = this.notas[index]  
    console.log(data)  
    this.$emit('editar', data)  
}
```

Este evento hará que se active en la vista `Notas.vue` una función para poder editar una nota, y también envía a la vista los datos (id, texto) de la nota que se está eligiendo.

Volvemos a `Notas.vue`, primero crearemos una pequeña estructura HTML donde editar la nota seleccionada:



```
<h2>
  Notas 📌
</h2>

<label for="editNote">Editar nota</label>

<input name="editNote" v-model="editTexto">

<button @click="updateNote()">
  ACTUALIZAR NOTA
</button>

<notas :notas="notas"
  v-on:editar="getNote">
</notas>
```

Añadiremos algunas variables nuevas en el HTML dentro de `data`:


```
data(){
  return {
    texto: '',
    editText: '',
    notas: [],
    id: null
  }
}
```

A continuación crearemos una variable que guarde los datos que le llegan desde el componente:

Guardará el texto en `editText`, e id en `id`.

```
getNote(data){

  console.log(data)
  // GUARDA EL TEXTO DE LA NOTA EN UNA VARIABLE
  this.editText = data.texto
  // GUARDA EL ID DE LA NOTA EN UNA VARIABLE
  this.id = data.id

}
```

Finalmente, creamos la llamada AXIOS para actualizar la nota, llamada `updateNote()`:

```
// CREANDO FUNCIÓN PARA ACTUALIZAR NOTA
updateNote(){
  var self = this
  // Pasamos el id de la nota como parámetro y lo sumamos
  // a la URL de la API
  axios.put('http://localhost:3050/notas/update/' + this.id, {
    texto: self.newText,
    id: self.id
  })
  .then(function (response) {
    console.log(response)
    // Actualizar la página
    location.reload()
  })
  .catch(function (error) {
    console.log(error)
  })
}
```

Asegúrate de haber enlazado la función `updateNote()` al botón de ACTUALIZAR en el HTML.

Ahora, si clicas '**EDITAR**' en alguna nota, editas su texto en el input superior y finalmente le das a **ACTUALIZAR**, el texto de la nota se actualizará y la página se actualizará automáticamente.

◆ 9. Borrando las notas con DELETE

La última operación es borrar las notas.

Para ello añadiremos dentro del componente `ShowNotas.vue` un nuevo botón de "**BORRAR**" que tendrá un evento que cogerá el index de la nota y enviará su id.

```
<div v-for="(nota, index) in notas"
  :key="nota.id"
  class="nota">
  {{ nota.texto }}
  <button @click="editNoteEvent(index)">
    EDITAR
  </button>
  <button @click="removeNoteEvent(index)">
    BORRAR
  </button>
</div>
```

El evento:

```
removeNoteEvent(index){
  let data = this.notas[index].id
  console.log(data)
  this.$emit('borrar', data)
}
```

En `api.js`, crearemos la lógica para borrar la nota:

```
// BORRANDO UNA NOTA
app.delete('/notas/del/:id', (req, res) => {
  const id = req.params.id

  const sql = `DELETE FROM lista_notas WHERE id=${id}`
  connection.query(sql, error => {
    if(error) throw error
    res.send('Nota borrada')
  })
})
```

Y finalmente, en `Notas.vue`, crearemos la función AXIOS que hace la llamada para borrar las notas:

```
deleteNotes(data){
  this.id = data
  axios.delete('http://localhost:3050/notas/del/' + this.id, {
    id: this.id
  })
  .then( function (response) {
    location.reload()
    console.log(response)
  })
  .catch( function (error) {
    console.log(error)
  })
}
```

Sólo queda enlazar esta función de borrar con el evento de borrar del componente:

```
<notas :notas="notas"  
  v-on:editar="showEditText"  
  v-on:borrar="deleteNotes">  
</notas>
```

Ahora, si intentas borrar una nota con el botón de BORRAR, se borrará de la base de datos.

◆ 10. Ejercicios



Implementa la librería SweetAlert2 para que indique a la persona usuaria cuándo se ha creado, editado y borrado una nota (con mensajes diferentes indicando qué acción se ha realizado).



Crea la lógica necesaria para recoger una nota con un `id` específico desde el front, hacer la llamada a través de la API a la base de datos y que el front muestre la información de dicha nota.



Implementa la librería vue-headful y cambia el título de las páginas de forma dinámica.



Cambia los atributos estáticos de router link (`to=""`) por atributos dinámicos (`:to=""`) para mejorar la navegación.



Crea una vista de Error o 404 para evitar que al visitar una URL que no existe en el proyecto, la página se vea en blanco.