

MINOTAUR



Juan Manuel Solano Alarcos



Índice

1. Introducción y justificación del proyecto.....	3
2. Antecedentes.....	4
3. Objetivos.....	5
4. Desarrollo práctico.....	5
a. Especificación de requisitos.....	5
b. Análisis y diseño.....	7
c. Implementación y pruebas.....	11
d. Resultados.....	35
e. Conclusiones.....	35
f. Nuevas propuestas.....	36
5. Anexos.....	37
a. Referencias bibliográficas.....	37
e. Código fuente.....	37



1. Introducción y justificación del proyecto

Mi objetivo con este proyecto es crear y documentar una aplicación de gestión de notas, tareas y eventos. La aplicación se llamará Minotaur, siendo este un juego de palabras mezclando “Mi” y “Nota” con el Minotauro de la mitología griega.

En un mundo donde la organización personal es esencial, la gestión adecuada de notas, tareas y eventos se ha convertido en una necesidad fundamental. El aumento de herramientas de organización lo refleja, sin embargo, muchas de estas herramientas no logran ser efectivas para los usuarios no expertos.

Minotaur surge como una solución diseñada para abordar estas necesidades mediante una aplicación intuitiva y multifuncional que centraliza la gestión de notas, tareas y eventos.

Las herramientas de productividad deben ser intuitivas y fáciles de usar para maximizar su efectividad. Minotaur se destaca por su diseño amigable y su facilidad de uso, permitiendo a los usuarios agregar, editar y organizar información con rapidez.

Minotaur no solo centraliza la información, sino que también se integra con otras aplicaciones y servicios populares (como calendarios y servicios de almacenamiento en la nube), garantizando una sincronización fluida y actualizada. Esta característica es crucial en un entorno donde los usuarios dependen de múltiples plataformas para gestionar diferentes aspectos de su vida personal y profesional.

Minotaur no es sólo otra aplicación de gestión de tareas; es una herramienta integral diseñada para satisfacer las complejas necesidades de organización de los usuarios modernos. Al ofrecer una plataforma centralizada, intuitiva y personalizable, Minotaur facilita una gestión eficiente y efectiva de notas, tareas y eventos, contribuyendo significativamente al aumento de la productividad y la organización personal. Con su enfoque en la integración, seguridad y accesibilidad.

2. Antecedentes

El campo de gestión de notas y eventos no es nada nuevo, existen numerosas aplicaciones con incontables herramientas. Algunos ejemplos serían: Notion, Obsidian, Logseq, Standard Notes y un largo etc...



Notion es una aplicación comercial con servicio gratuito para usuarios individuales que ofrece muchas herramientas de gestión.

Sin embargo, los archivos no se guardan localmente.

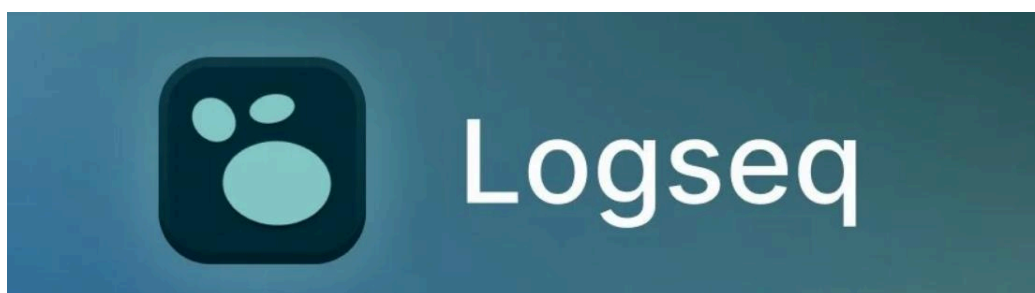
Obsidian es una aplicación más centrada en la toma de notas.

Utiliza Markdown y permite realizar conexiones entre las notas que más tarde se pueden visualizar mediante grafos.



Obsidian

Sin embargo, es una aplicación comercial y además no es de código abierto.



Logseq es una aplicación parecida a Obsidian, con conexión de grafos y uso de Markdown y OrgMode. De código abierto y no comercial, se centra en la seguridad y cuenta con un soporte de plugins. Sin embargo, no es una aplicación sencilla e intuitiva de usar.

El objetivo de Minotaur es ofrecer algo sencillo, no demasiado alejado del por todos conocido “Bloc de Notas” de Windows, pero aportando nuevas y mejores funcionalidades. Queremos **sencillez, rapidez, y transportabilidad**.



3. Objetivos

Este proyecto busca:

- Crear una aplicación para crear y gestionar notas.
- Ofrecer una alternativa simple, intuitiva, gratuita y de código abierto.
- Aprender sobre el desarrollo de aplicaciones de escritorio

4. Desarrollo práctico

a. Especificación de requisitos

REQUISITOS DEL SISTEMA

REQUISITOS MÍNIMOS:

- Sistema operativo: Windows 7. Ubuntu 18. Debian 10.
- Procesador: Intel Dual Core a 2 GHz o similar.
- Memoria: 4 GB.
- Tarjeta gráfica: Gráficos integrados del procesador.
- Almacenamiento: 100 MB de espacio disponible.

REQUISITOS RECOMENDADOS:

- Sistema operativo: Windows 10/11 de 64 bits. Ubuntu 22/24. Debian 12.
- Procesador: Intel Core i3 o similar.
- Memoria: 8 GB o más.
- Tarjeta gráfica: Gráficos integrados del procesador.
- Almacenamiento: 250 MB de espacio disponible.

REQUISITOS FUNCIONALES

- Herramientas: Los usuarios podrán crear notas, tareas y eventos. Las notas se creará mediante un editor integrado en la aplicación teniendo un título y un cuerpo que más tarde se guardarán. Las tareas se añadirán en otra pestaña dónde se podrán separar por categorías en función de su progreso. Por último los eventos estarán hecho a través de la API de Google con Google Calendar, pudiendo añadir eventos en la misma aplicación.



- Navegación de carpetas: Para que los usuarios tengan una experiencia más fluida al organizar sus notas, habrá un buscador integrado que enseñará todos los subdirectorios y archivos de un directorio que el usuario elija. Además podrá crear nuevas carpetas para una mejor organización de las notas o eliminarlas. Los directorios se actualizan solos pero en caso de error podremos recargar. Por último se podrá abrir otros directorios que queramos además de otros archivos, sin necesidad de cambiar nuestra configuración.
- Markdown: Las notas se guardarán en formato .md y tendremos la opción de escribir con markdown y alternar el modo de visualización con un botón para ver cómo va quedando.
- Herramientas de notas: El usuario contará con un par de herramientas que harán más sencilla la creación de notas. Entre ellas estarán el botón de limpiar nota, que dejaría en blanco la nota. El botón de nueva nota para abrir otra nota, un indicador de guardado para ver si la nota está guardada, dos botones de guardado, uno en la nube y por último dos botones, uno para cifrar y otro para descifrar la nota.
- Guardado: Para guardar una nota se presionará en el botón de guardado, esté comprueba si la nota está guardada en el sistema de archivos. En caso de estarlo se guardan sin más, actualizando los valores. Sin embargo, si es una nueva nota, nos saldrá una ventana para seleccionar dónde guardar la nota.
- Configuración por defecto: Al entrar a la aplicación por primera vez habrá que configurarla. Hay que elegir un directorio por defecto y una clave de cifrado. También se podrá cambiar la paleta de colores.
- Alta de usuario: Los clientes podrán crear usuarios locales o entrar con uno anónimo.
- Login: Los usuarios locales estarán protegidos por contraseña.
- Transportabilidad: Los usuarios podrán guardar archivos en la nube a través de la API de Google con Google Drive.
- Seguridad: El sistema encriptará los archivos que se marquen como sensibles pudiendo revertir el proceso si se conoce la clave de cifrado.
- Responsivo: El sistema creará un evento en el calendario al indicar la fecha.



REQUISITOS NO FUNCIONALES

- Confianza: La aplicación será de código abierto y estará subida a github de forma pública.
- Sincronización: El usuario podrá conectarse con Google Drive y así guardar sus notas en la nube.
- Personalización: La interfaz será personalizable. Pudiendo así cambiar la paleta de colores
- Backup: Se creará una archivo temporal a modo de backup en caso de fallar la aplicación.
- Seguridad: Los usuarios locales podrán guardar sus archivos encriptados. Esto se hará mediante una clave que asegure que solo el usuario tiene acceso a sus notas.

REQUISITOS DE DESARROLLO

IDE: Visual Studio Code 1.88.1.

Framework de aplicación: Electronjs 30.0.0.

Lenguajes de programación usados:

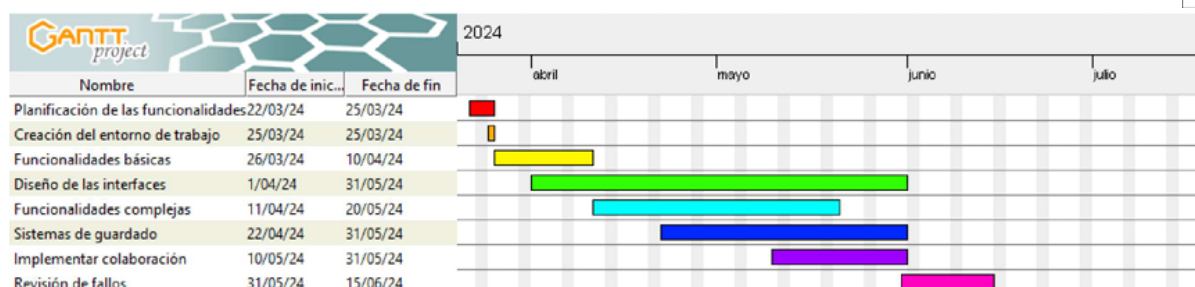
- NodeJS v20.11.
- JavaScript ES2024 con React.
- CSS 3 con Bulma.
- HTML 5.
- Markdown 1.0.1.
- Marked y zero-md (parser de Markdown)

Controlador de versiones: Git y Github.

b. Análisis y diseño

TEMPORALIZACIÓN

Diagrama de Gantt



MODELOS Y DIAGRAMAS

Wireframes

- NOTAS





- EVENTOS

Minotaur

Notas Eventos Calendario

TOOLS

Item 1

Item 2

Item 3

Item 4

Item 5

☒ EVENTO 1

☒ EVENTO 2

☐ EVENTO 3

☐ EVENTO 4

☒ EVENTO 5

☐ EVENTO 6



- CALENDARIO

Minotaur

Notas

Eventos

Calendario

TOOLS

EVENTOS PLANEADOS del día

10

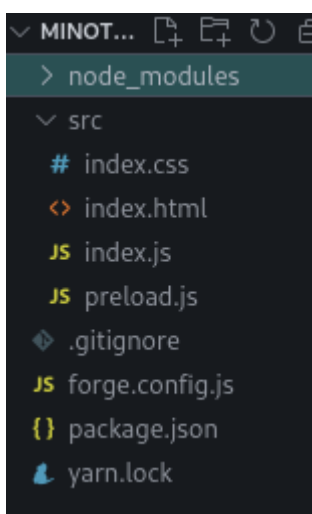
c. Implementación y pruebas

PREPARACIÓN DEL ENTORNO DE DESARROLLO

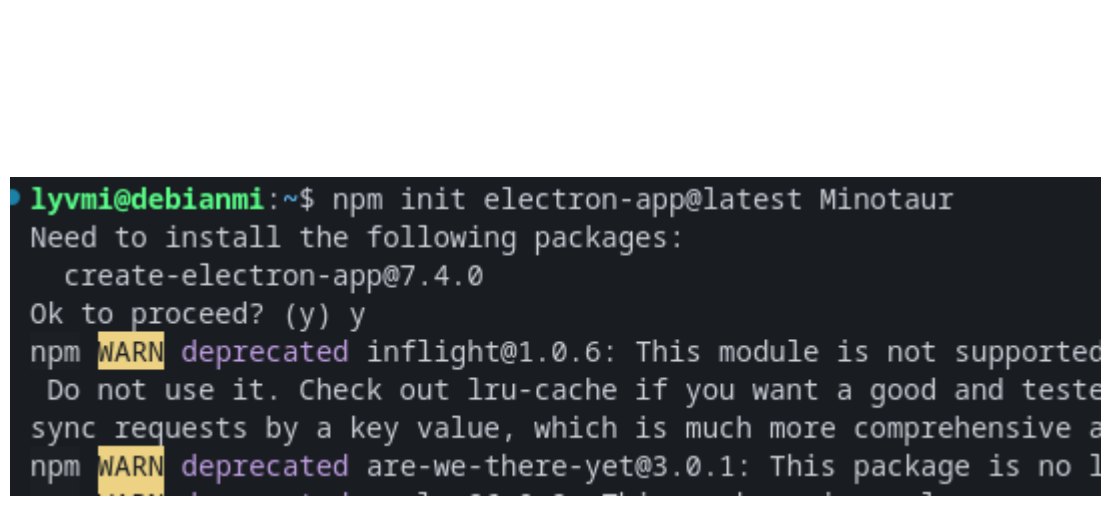
Se hará uso de un entorno de Linux Debian para la preparación. Necesitamos tener instalado Visual Studio Code.

PASOS

Para preparar el entorno de Electron voy a usar un Electron Forge que creará un entorno preconfigurado y nos ayudará a empezar más rápido.



```
MINOT... [Icons]
├── node_modules
├── src
│   ├── # index.css
│   ├── <> index.html
│   ├── JS index.js
│   ├── JS preload.js
│   ├── .gitignore
│   ├── JS forge.config.js
│   ├── {} package.json
│   └── yarn.lock
└──
```



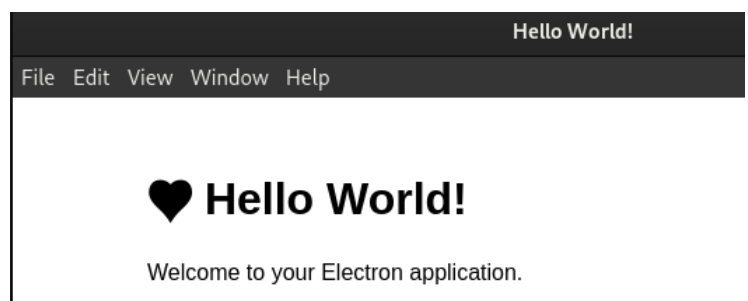
```
lyvmi@debianmi:~$ npm init electron-app@latest Minotaur
Need to install the following packages:
  create-electron-app@7.4.0
Ok to proceed? (y) y
npm WARN deprecated inflight@1.0.6: This module is not supported
Do not use it. Check out lru-cache if you want a good and tested
sync requests by a key value, which is much more comprehensive a
npm WARN deprecated are-we-there-yet@3.0.1: This package is no l
```

La podemos ejecutar para ver la plantilla que nos ha dejado

```
lyvmi@debianmi:~/Minotaur$ npm start

> minotaur@1.0.0 start
> electron-forge start

✓ Checking your system
✓ Locating application
✓ Loading configuration
```





Y comprobamos que ya tenemos una ventana con la aplicación. Vamos a la carpeta de src y explicar qué tenemos y cómo funciona Electron.

Electron construye aplicaciones haciendo que se ejecuten como si de un navegador se tratara. Para ello empaqueta Chromium con la aplicación, lo cual lo hace un poco pesado pero compatible con la gran mayoría de plataformas.

Debido a que Electron funciona parecido a un navegador, este usa html, css y javascript para dar forma y utilidad a sus elementos. Y con la ayuda de Node.js podemos utilizar las funciones de nuestro sistema como, por ejemplo, para guardar archivos.

Más adelante profundizaremos en cómo se renderizan las ventanas y cómo hacemos uso del sistema.

CSS

```
# index.css U X  <> index.html U
src > # index.css > ...
1  body {
2    font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI',
    Roboto, Helvetica,
3    Arial, sans-serif;
4    margin: auto;
5    max-width: 38rem;
6    padding: 2rem;
7  }
```

HTML

```
index.css U  <> index.html U X
<> index.html > ...
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Hello World!</title>
    <link rel="stylesheet" href="index.css" />
  </head>
  <body>
    <h1>♥ Hello World!</h1>
    <p>Welcome to your Electron application.</p>
  </body>
</html>
```



index.js

```
index.js 7 ...
const { app, BrowserWindow } = require('electron');
const path = require('node:path');

// Handle creating/removing shortcuts on Windows when installing/uninstalling.
if (require('electron-squirrel-startup')) {
  app.quit();
}

const createWindow = () => {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      preload: path.join(__dirname, 'preload.js'),
    },
  });

  // and load the index.html of the app.
  mainWindow.loadFile(path.join(__dirname, 'index.html'));
```

Encargado de renderizar la ventana y los elementos de la aplicación.

preload.js

```
# index.css U  JS index.js U  JS preload.js U X
src > JS preload.js
1 // See the Electron documentation for details on how to use preload s
2 // https://www.electronjs.org/docs/latest/tutorial/process-model#prel
3
```

El preload se encargará de precargar las funciones, en mi caso no la usaré.

Ahora vamos a instalar Bulma CSS para nuestro proyecto.

Hay diferentes formas de hacerlo, pero yo he optado por poner el enlace a una url.

```
<meta name="viewport" content="width=device-width,
initial-scale=1.0">
<link
rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bulma@1.0.0/css/bulma.min.css"
>
```



También voy a añadir un link para añadir unos iconos al proyecto, llamados Boxicons, que son gratuitos.

```
css >  
<link href='https://unpkg.com/boxicons@2.1.4/css/boxicons.min.css' rel='stylesheet'>
```

Ahora vamos a crear nuestra configuración de archivos. Debido a que voy a implementar 3 herramientas y un apartado de opciones, voy a dividir los HTML, CSS y JavaScripts en 4.

```
▼ HTML-CSS  
# calendar.css  
<> calendar.html  
# index.css  
<> index.html  
# options.css  
<> options.html  
# task.css  
<> task.html  
> img  
JS calendar.js  
JS index.js  
JS options.js  
JS preload.js  
JS renderer.js  
JS task.js
```

Cada herramienta tendrá un HTML y un CSS adherido, además vinculamos el js de cada opción.

En el caso de index.html (que sería la herramienta notas, la principal), van asociados con él “index.css” y “renderer.js”.

Ya por último aclarar que lo he organizado en carpetas separadas y he añadido una de imágenes que posteriormente necesitaremos.

Hasta aquí la preparación del entorno de desarrollo.

INSTALACIÓN DE PARSER DE MARKDOWN

Ahora vamos a instalar una dependencia de nuestro proyecto, “Marked”. Esto lo que hará será convertir el html sin formato de la nota y aplicar el formato de Markdown.

Metemos este comando en la consola de nuestro proyecto:

```
● lyvmi@debianmi:~/Minotaur$ npm install marked  
  
added 1 package, and audited 483 packages in 737ms  
  
76 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities  
○ lyvmi@debianmi:~/Minotaur$
```



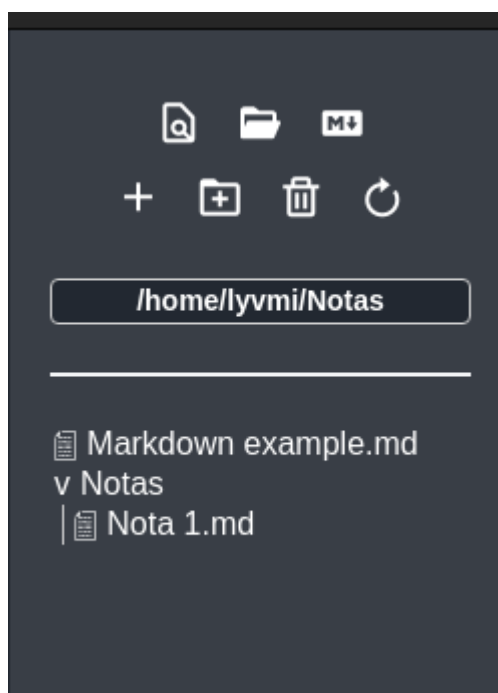

La barra de navegación es un elemento que se encontrará en todas las herramientas menos en opciones, que tendrá otra navegación más adaptada para la configuración.

EXPLORADOR DE CARPETAS

En notas habrá un explorador para poder elegir las notas y organizarlas. Está cuenta con un par de herramientas. y tendría la siguiente estructura.

```
<div class="folder-container">
  <div class="folder-menu">
    <header class="folder-buttons">
      <i class='bx bx-file-find' title="Cargar fichero"></i>
      <i class='bx bxs-folder-open' title="Cargar directorio"></i>
      <i class='bx bxl-markdown' title="Modo markdown"></i>
    </header>
    <footer class="folder-options">
      <i id="new-note" class='bx bx-plus' title="Crear nueva nota"></i>
      <i class='bx bx-folder-plus' title="Crear nuevo directorio"></i>
      <i class='bx bx-trash' title="Modo eliminar"></i>
      <i id="reload" class='bx bx-revision' title="Recargar"></i>
    </footer>
    <div id="current-path"></div>
  </div>
  <hr>
  <div class="folder-tree"></div>
</div>
```

Que se vería así:



Las herramientas aparecen arriba, después el directorio actual y por último el árbol de carpetas.



Para el correcto funcionamiento de abrir y cerrar el explorador al pulsar en el interruptor de la barra lateral, he usado este javascript, que cambia las clases para que después el css haga su trabajo.

```
toggle.addEventListener("click", () => {
  folder_container.classList.toggle("close");
  toggle.classList.toggle("close");
  note_container.classList.toggle("close");

  if (toggle.classList.contains("bx-chevron-right")) {
    toggle.classList.remove("bx-chevron-right");
    toggle.classList.add("bx-chevron-left");
  } else {
    toggle.classList.remove("bx-chevron-left");
    toggle.classList.add("bx-chevron-right");
  }
});
```

Para conseguir los directorios y archivos he creado dos funciones fundamentales. La primera crea los elementos y la segunda los enseña.

```
// Function to generate directory tree HTML
> function buildDirectoryTreeHTML(directoryPath, callback) { ...
}

// Function to display directory contents
> function displayDirectoryContents(directoryPath) { ...
}
```

displayDirectoryContents llama a buildDirectoryTreeHTML que lee de nuestro sistema de archivos las carpetas y ficheros y crea una estructura de elementos con ellos. Esta estructura se añadirá posteriormente al HTML, en el <div class="folder-tree"></div>.

Además, se le añaden Event Listeners a los elementos que se usarán para abrir los ficheros, o desplegar las carpetas o para abrir un menú contextual.

```
// Initial display
displayDirectoryContents(notes_directory);
```

La inicialización al abrir la aplicación.

HERRAMIENTAS DEL EXPLORADOR

El explorador tiene 7 botones diferentes. Arriba están el de abrir una nota, abrir otro directorio y cambiar el modo de markdown.

Abrir una nota, abre una ventana nueva para seleccionar la nota que quieras abrir aunque no esté en el directorio que tengas abierto.

```
function openFileDialog() {
  ipcRenderer.invoke('show-open-file-dialog').then(result => {
    if (!result.canceled) {
      openItem(result.filePaths[0], true);
    }
  }).catch(err => {
    console.error('Error opening file dialog:', err);
  });
}
```

Abrir un directorio abrirá un nuevo directorio diferente al de por defecto, aunque este no se mantendrá abierto después de salir de notas.

```
function openFolderDialog() {
  ipcRenderer.invoke('show-open-folder-dialog').then(result => {
    if (!result.canceled) {
      notes_directory = result.filePaths[0];
      displayDirectoryContents(notes_directory);
    }
  }).catch(err => {
    console.error('Error opening folder dialog:', err);
  });
}
```

Después el modo markdown lo que hace es alternar el modo de visualización de la nota, entre markdown y texto plano. En el modo markdown no se puede escribir por lo que habrá que volver a alternar de modo.

```
markdown_toggle.addEventListener("click", () => {
  markdown_toggle.classList.toggle("active");
  if (markdown_toggle.classList.contains("active")) {
    text = body.querySelector(".note-body").value;
    md_text.innerHTML = text;
    noteBody.style.display = "none";
    zero_md.style.display = "block";
    markdown_toggle.classList.remove("bxl-markdown");
    markdown_toggle.classList.add("bx-edit-alt");
    markdown_toggle.title = "Modo editar";
  } else {
    noteBody.style.display = "block";
    zero_md.style.display = "none";
    markdown_toggle.classList.add("bxl-markdown");
    markdown_toggle.classList.remove("bx-edit-alt");
    markdown_toggle.title = "Modo markdown";
  }
});
```



Las siguientes herramientas serían: crear una nueva nota, crear una nueva carpeta, modo eliminar y recargar.

Las dos primeras se explican solas. Crear una nueva nota solamente limpia el todo el texto de las notas, crear una nueva carpeta es más complicado. Primero se crea un cuadro de texto en el que pondrás el nombre de la carpeta. Si le das al enter o quitas el focus de la carpeta esta se creará. También puedes crear una carpeta dentro de otra haciendo click derecho en la carpeta y dándole a crear nueva carpeta.

```
function nameNewFolder(folderpath) {
  const folder_name = document.createElement("input");
  let sent = false;
  let folder;
  folder_name.type = "text";
  folder_name.classList = "folder";
  if (folderpath) {
    folder = document.querySelector(`[data-filepath="${folderpath}"]`);

    if (!folder.classList.contains("show")) {
      folder.classList.add("show")
    }
    folder.appendChild(folder_name);
  }
  else {
    folder_tree.appendChild(folder_name);
  }
  folder_name.focus();
  folder_name.addEventListener("focusout", (e) => {
    if (!sent) {
      const newFolderName = folder_name.value;
      if (newFolderName.includes(".") || !newFolderName) {
        console.log("Name can't contain '.' or be null");
        folder_name.remove();
        sent = true
      }
      else {
        let newFolderPath;
        if (folder) {
          newFolderPath = path.join(folderpath, newFolderName);
        }
        else {
          newFolderPath = path.join(notes_directory, newFolderName);
        }
      }
    }
  });
}
```

El modo eliminar permite eliminar notas o carpetas haciendo doble click sobre ellos. Este modo no se desactiva hasta que le vuelvas a dar al botón. Alternativamente puedes borrar elementos haciendo clic derecho.

```
function deleteItem(filePath) {
  fs.stat(filePath, (err, stats) => {
    let element_name = filePath.split("/");
    element_name = element_name.slice(-1);
    if (err) {
      console.error('Error accessing file/directory:', err);
      return;
    }

    if (stats.isDirectory()) {
      let confirm = window.confirm('¿Seguro que quieres borrar el directorio ' + element_name + '?'');
      if (confirm) {
        fs.rmdir(filePath, { recursive: true }, (err) => {
          if (err) {
            console.error('Error deleting directory:', err);
            return;
          }
          console.log('Directory deleted successfully:', filePath);
          displayDirectoryContents(notes_directory);
        });
      }
    } else {
      let confirm = window.confirm('¿Seguro que quieres borrar la nota "' + element_name + '"?');
      if (confirm) {
        fs.unlink(filePath, (err) => {
          if (err) {
            console.error('Error deleting file:', err);
          }
        });
      }
    }
  });
}
```

También te avisa antes de borrar la carpeta.

Por último recargar, que solo actualiza los directorios, en caso de que no se hayan actualizado solos. Por lo general los directorios se actualizan solos.



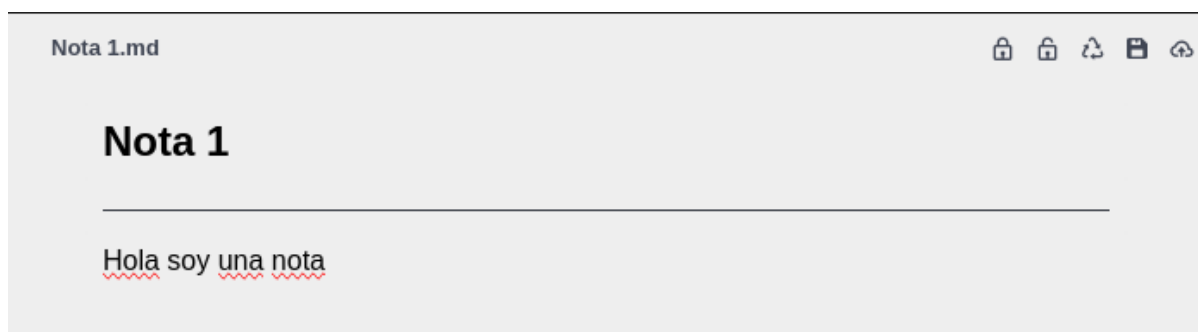
EDITOR DE NOTAS

Ahora llegamos al editor de notas, dónde pasaremos gran parte del tiempo. La estructura es la siguiente:

```
<!-- NOTE CONTAINER -->
<div class="note-container">
  <nav class="tool-bar">
    <p class="note-name"></p>
    <i class='bx bx-lock encrypt' title="Encriptar"></i>
    <i class='bx bx-lock-open decrypt' title="Desencriptar"></i>
    <i class='bx bx-recycle' title="Limpiar nota"></i>
    <i class='bx bxs-save' title="Guardar"></i>
    <i class='bx bx-cloud-upload' title="Guardar en la nube"></i>
  </nav>
  <div class="note-editor">
    <input type="text" class="note-title" placeholder="Title">
    <hr>
    <textarea class="note-body" placeholder="Write your note here..."></textarea>
    <zero-md class="zero-md" style="display: none;">
      <template>
        <style>
          body{
            background-color: #EEEEEE;
          }
        </style>
      </template>
      <script type="text/markdown" class="md-text">

    </script>
  </template>
</zero-md>
</div>
</div>
```

Aquí podemos ver las herramientas arriba y el editor debajo, con un título para la nota y un cuerpo. Se vería así:



Una interfaz sencilla y elegante.

FUNCIONES DE NOTAS

El editor de notas cuenta con 5 opciones. Una de cifrar, descifrar, limpiar la nota, guardar localmente y guardar en la nube.

```
function encryptNote(key, text) {
  if (isNoteOpened) {
    const cipher = crypto.createCipher('aes-256-cbc', key);
    let encryptedText = cipher.update(text, 'utf8', 'hex');
    encryptedText += cipher.final('hex');
    return encryptedText;
  }
  else{
    return "";
  }
}

function decryptNote(key, encryptedText) {
  try {
    const decipher = crypto.createDecipher('aes-256-cbc', key);
    let decryptedText = decipher.update(encryptedText, 'hex', 'utf8');
    decryptedText += decipher.final('utf8');
    return decryptedText;
  } catch (error) {
    return "" // Return null or handle the error appropriately
  }
}
```

Se cifra y se descifra con la clave que hayamos puesto en la configuración inicial

Limpiar la nota solamente borra todo lo que haya escrito en la nota, pero seguiremos editando esa nota.

Y por último el guardado,

```
function SaveNote(encBody) {
  const title = noteTitle.value;
  let note_body = noteBody.value;
  if (encBody) {
    note_body = encBody;
  }
  if (!isNoteOpened) {
    ipcRenderer.send('save-note', { title: title, body: note_body, defaultRootDirectory: notes_directory })
  } else {
    const file = document.querySelector(`[data-filepath="${savedfilepath}"]`)
    const filePath = path.join(file.dataset.filepath);
    fs.writeFile(filePath, `${title}-----${note_body}`, (err) => {
      if (err) {
        console.error('Error saving file:', err);
      } else {
        console.log('File saved successfully:', filePath);
        savedTitle = noteTitle.value;
        savedBody = noteBody.value;
        noteName.classList.remove("notSaved");
      }
    })
  }
}
```



El guardado funciona en dos partes. Primero comprueba si es una nueva nota, en tal caso envía un mensaje al proceso principal que se encargará de abrir una ventana nueva con un explorador de archivos para decir dónde queremos guardar la nota. Y una vez tenga la ruta guardará la nota.

```
ipcMain.on('save-note', (event, noteData) => {
  const title = noteData.title;
  const body = noteData.body;
  const defaultRootDirectory = noteData.defaultRootDirectory;

  if (isDialogOpen) {
    return;
  }

  isDialogOpen = true;

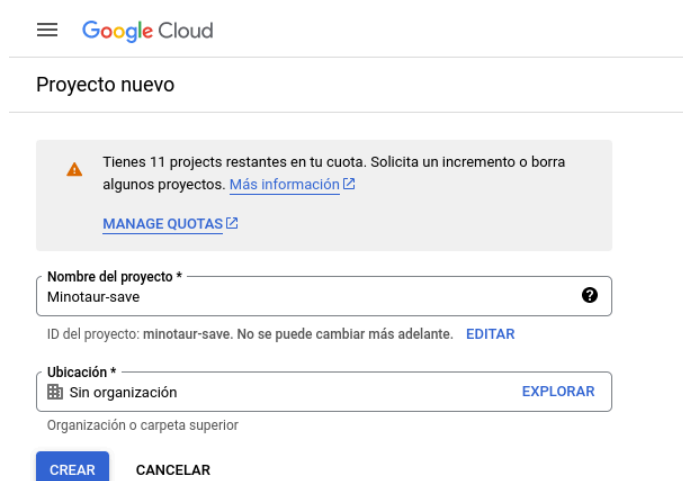
  dialog.showSaveDialog({
    title: 'Save Note',
    defaultPath: path.join(defaultRootDirectory, `${title}.md`),
    filters: [
      { name: 'Text Files', extensions: ['md', 'txt'] }
    ]
  }).then(result => {
    isDialogOpen = false;
  });
});
```

```
fs.writeFile(filePath, `${title}---...---.---${body}`, async (err) => {
```

Se guarda con esa ristra de caracteres “-” y “.” para codificar el título y el cuerpo en la misma línea.

GUARDADO EN LA NUBE

Para el guardado en la nube voy a usar la API de google y así guardar los archivos en el Drive. Para ello tenemos que ir a Google Cloud Console.



Google Cloud

Proyecto nuevo

Tienes 11 projects restantes en tu cuota. Solicita un incremento o borra algunos proyectos. [Más información](#)

[MANAGE QUOTAS](#)

Nombre del proyecto *

Minotaur-save

ID del proyecto: minotaur-save. No se puede cambiar más adelante. [EDITAR](#)

Ubicación *

Sin organización [EXPLORAR](#)

Organización o carpeta superior

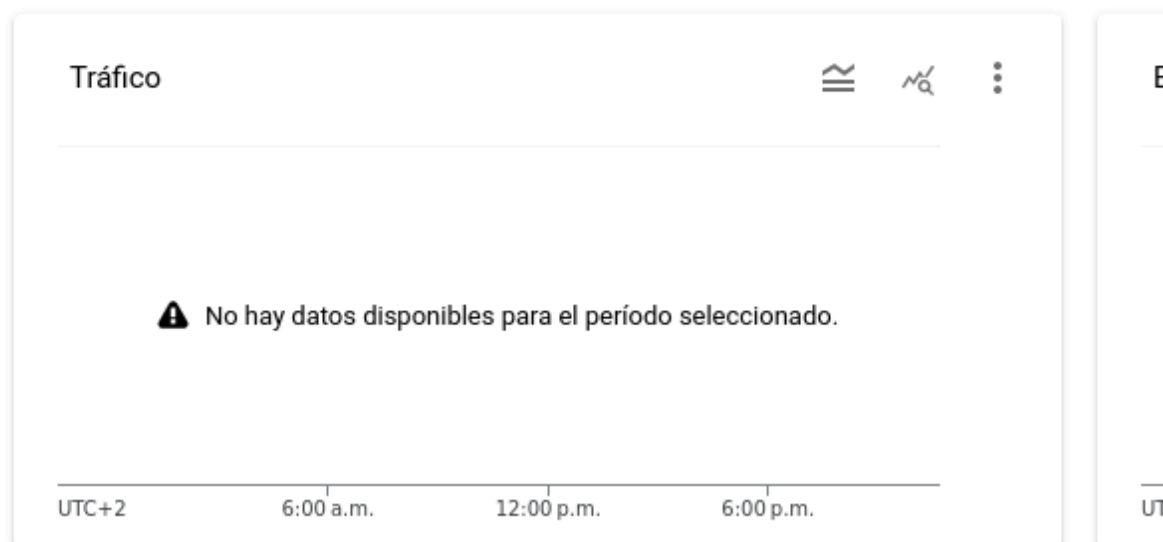
[CREAR](#) [CANCELAR](#)



Creamos un nuevo proyecto. Después vamos a APIs y servicios dónde habilitaremos nuestras APIs.

APIs y servicios

[+ HABILITAR APIS Y SERVICIOS](#)



[Filtro](#) Filtro

Nombre	↓ Solicitudes	Errores (%)	Latencia mediana (ms)
Google Calendar API			
Google Drive API			

Después vamos a credenciales y añadimos un de ID de Clientes OAuth

ID de clientes OAuth 2.0

<input type="checkbox"/>	Nombre	Fecha de creación ↓
<input type="checkbox"/>	Minotaur	9 jun 2024

Tendremos que también crear una hoja de consentimiento para añadir nuestros servicios y permisos que va a tener nuestra aplicación con google y sus usuarios.



OAuth consent screen

Minotaur EDIT APP

Publishing status

Testing

PUBLISH APP

User type

External 

MAKE INTERNAL

OAuth user cap

Una vez hecho eso nos descargamos las credenciales y nos darán los datos necesarios para implementar la API:

```
{ "installed": { "client_id": "454455339224-0ejfctpl1a6u1nau00hqlvskrvpu4t.apps.googleusercontent.com", "project_id": "minotaur-save", "auth_uri": "https://accounts.google.com/o/oauth2/auth", "token_uri": "https://oauth2.googleapis.com/token", "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs", "client_secret": "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", "redirect_uris": [ "http://localhost" ] }
```

Ahora en nuestra aplicación tendríamos que crear un js que se encargue de autenticar la conexión y conectarse con el servicio

```
const oauth2Client = new google.auth.OAuth2(
  CLIENT_ID,
  CLIENT_SECRET,
  REDIRECT_URI
);

// Scopes for Google Drive
const SCOPES = ['https://www.googleapis.com/auth/drive.file'];

function getAuthUrl() {
  return oauth2Client.generateAuthUrl({
    access_type: 'offline',
    scope: SCOPES,
  });
}

async function authenticate(code) {
  const { tokens } = await oauth2Client.getToken(code);
  oauth2Client.setCredentials(tokens);
}
```



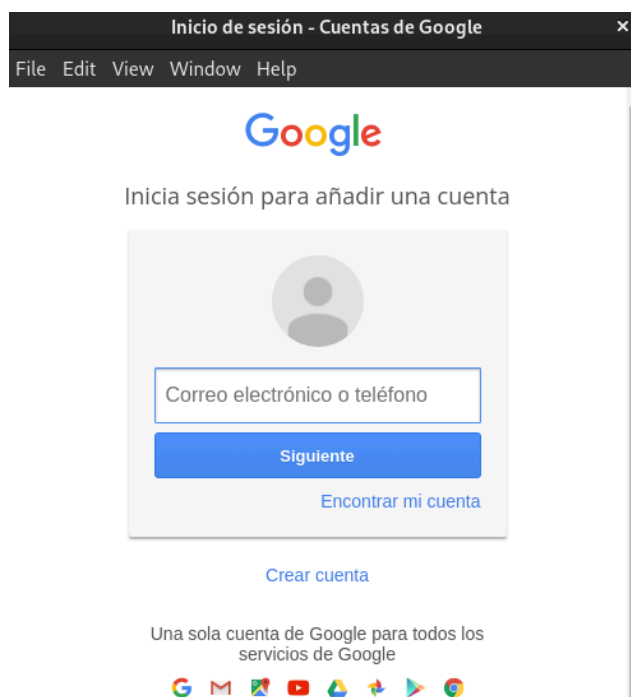
Y luego en el index.js decirle que cree una ventana con los datos para autenticarse cuando pulsemos el botón de guardar en la nube.

```
ipcMain.handle('google-authenticate', async (event) => {
  const authUrl = getAuthUrl();
  const authWindow = new BrowserWindow({
    width: 500,
    height: 600,
    webPreferences: {
      nodeIntegration: false,
      contextIsolation: true,
    }
  });

  authWindow.loadURL(authUrl);

  authWindow.webContents.on('will-redirect', async (event, url) => {
    if (url.startsWith(REDIRECT_URI)) {
      const urlObj = new URL(url);
      const code = urlObj.searchParams.get('code');
      try {

```



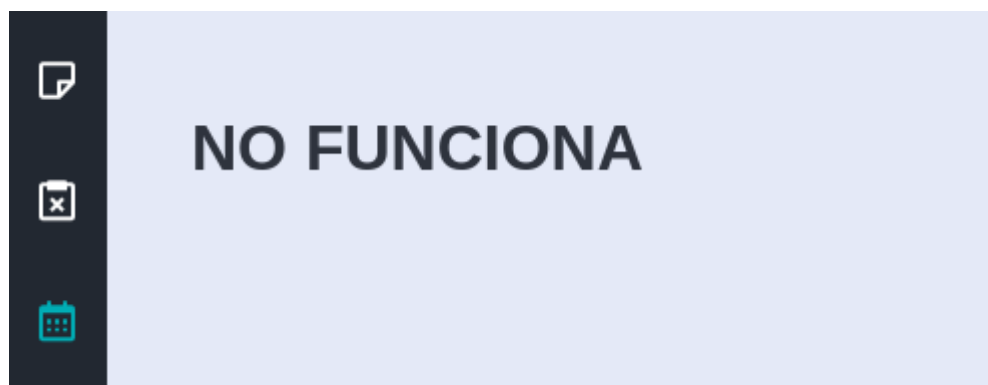
Y funciona, ahora nos conectaremos y podríamos guardar y descargar nuestros ficheros



Sin embargo... no he conseguido que funcione la autenticación. Dejaré el código en la versión final, ya que afecta al resto de la aplicación.

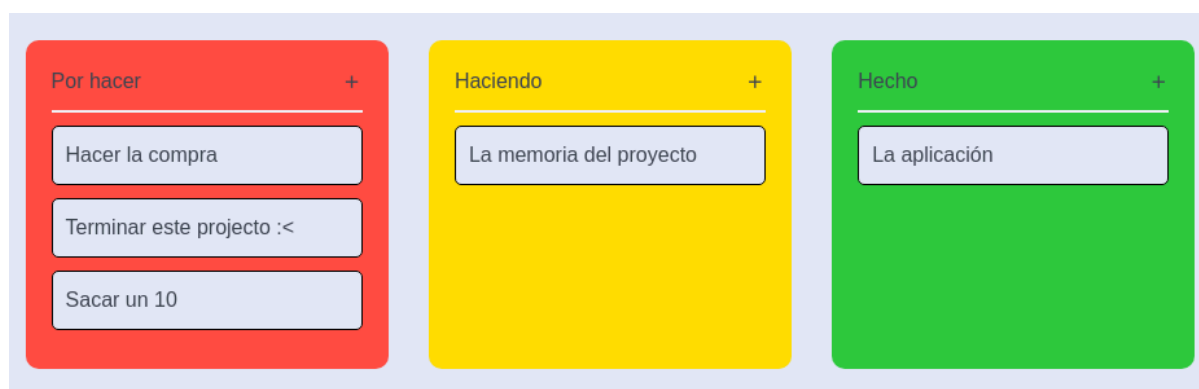
CALENDARIO

Esto significa que el calendario tampoco lo hará, porque utilizaba también la API de google para conectarse a google calendar.



TAREAS

Las tareas por otra parte, si funcionan muy bien. Podemos crear tareas y organizarlas en tres bloques, por hacer, haciendo y hechas.



También podemos crear tareas dándole al “+” y ponerle un nombre:



Añadir tarea

Nombre de la tarea

Añadir Cancelar

Y no solo eso, sino que podemos moverlas entre ellas con solo arrastrar las tareas.

Por hacer +

Hacer la compra

Haciendo +

La memoria del proyecto

Terminar este proyecto :<

Hecho +

La aplicación

Sacar un 10

Por supuesto también se pueden borrar, para ello hacemos click derecho en la nota que queramos borrar y nos saldrá un menú dónde podremos darle a “Eliminar”.

Hecho +

La aplicación

Eliminar

Las tareas se conservan entre sesiones, para ello se guardan en un .json divididas por bloques.

```
{
  "todo": [
    {
      "id": "1717986865225",
      "text": "Hacer la compra"
    },
    {
      "id": "1717986880882",
      "text": "Sacar un 10"
    },
    {
      "id": "1717986875382",
      "text": "Terminar este proyecto :<"
    }
  ],
  "doing": [
    {
      "id": "1717986890928",
      "text": "La memoria del proyecto"
    }
  ],
  "done": [
    {
      "id": "1717986903354",
      "text": "La aplicación"
    }
  ]
}
```

Cada tarea tiene asignado su propio id, esto nos permite distinguirlas unas de otras.

Muy útil en la función de eliminar.

El id de la tarea es su fecha de creación en unix time.

```
// Save tasks to JSON
function saveTasks() {
  const homeDirectory = os.homedir();
  const tasksDir = path.join(homeDirectory, '.minotaur');
  const filePath = path.join(tasksDir, 'tasks.json');
  fs.writeFileSync(filePath, JSON.stringify(tasks, null, 2));
}

function deleteTask(taskId, column) {
  const index = tasks[column].findIndex(task => task.id === taskId);
  if (index !== -1) {
    tasks[column].splice(index, 1);
    renderTasks();
    saveTasks();
  }
}
```

Ya para terminar, la forma en la que se pueden mover las tareas es un tanto compleja. Pero se resume en obtener las coordenadas x e y del elemento más cercano e insertar la nota delante o detrás. Y por supuesto usar los eventos de drag.

```
// Get the element after which the task should be inserted
function getDragAfterElement(container, y) {
  const draggableElements = [...container.querySelectorAll('.task:not(.dragging)')];

  return draggableElements.reduce((closest, child) => {
    const box = child.getBoundingClientRect();
    const offset = y - box.top - box.height / 2;
    if (offset < 0 && offset > closest.offset) {
      return { offset: offset, element: child };
    } else {
      return closest;
    }
  }, { offset: Number.NEGATIVE_INFINITY }).element;
}
```

```
function addDragAndDrop() {
  const tasks = document.querySelectorAll('.task');
  const columns = document.querySelectorAll('.tasks');

  tasks.forEach(task => {
    task.addEventListener('dragstart', () => {
      task.classList.add('dragging');
    });

    task.addEventListener('dragend', () => {
      task.classList.remove('dragging');
      updateTasks();
    });
  });

  columns.forEach(column => {
    column.addEventListener('dragover', e => {
      e.preventDefault();
      const afterElement = getDragAfterElement(column, e.clientY);
      const draggingTask = document.querySelector('.dragging');
      if (afterElement == null) {
        column.appendChild(draggingTask);
      } else {
        column.insertBefore(draggingTask, afterElement);
      }
    });
  });
}
```

Como se ve, cada elemento tiene su propio event listener con un evento drag

OPCIONES

Ahora vienen las opciones que nos permitirá modificar la configuración de nuestra aplicación.

OPCIONES

General

Apariencia

General

Directorio predeterminado

Este será el directorio de notas que se visualizará por defecto. Buscar

Clave de cifrado

Esta clave se usará para cifrar tus notas, no la olvides.



Esta será la pestaña que primero veremos al entrar por primera vez. Nos obliga a definir un directorio por defecto que será el que veamos al abrir la aplicación de notas. Como vimos previamente este se podrá cambiar desde la propia herramienta de notas, pero se reiniciaría al por defecto cada vez que salgamos de la herramienta.

Y la clave de cifrado es la que usaremos para cifrar las notas. Puede estar vacía pero eso sería poco seguro.

Estas dos opciones se guardan en un json situado en el home del usuario, dentro de una carpeta llamada .minotaur

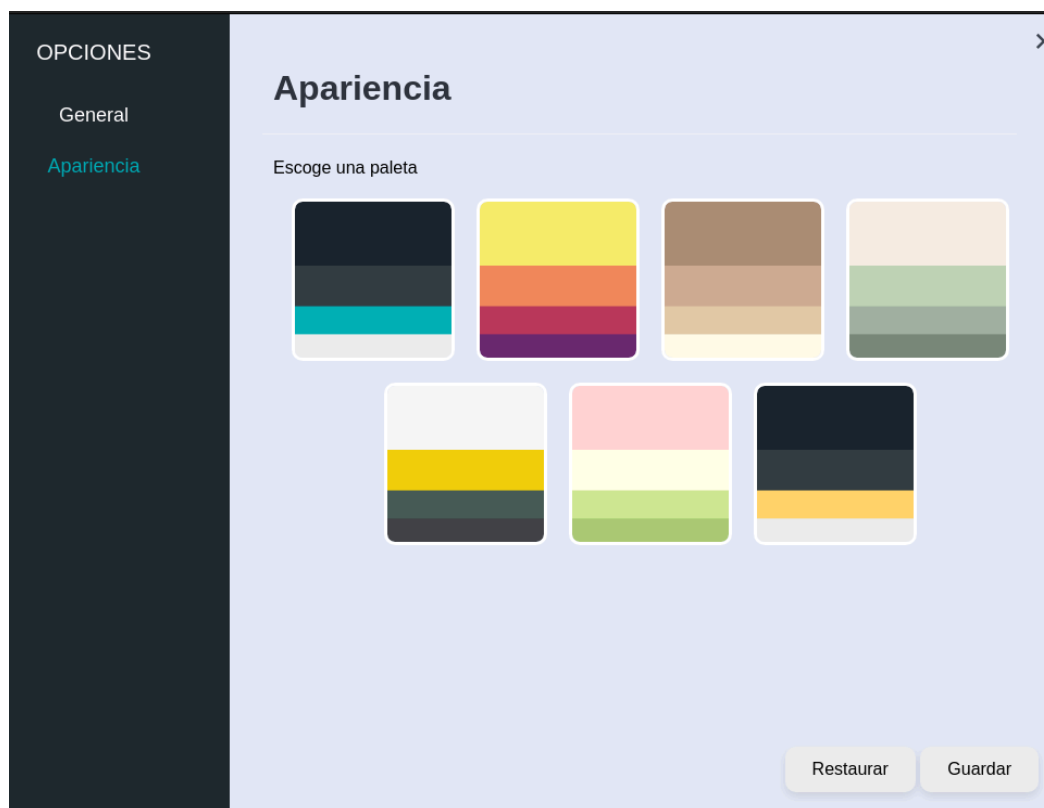
```
// Event listener for the general save button
generalSaveButton.addEventListener('click', () => {
  const defaultDirectory = document.querySelector('#default-directory').value;
  const encryptionKey = document.querySelector('#encryption-key').value;
  const palette = selectedPalette;

  const configData = {
    defaultDirectory,
    encryptionKey,
    palette
  };

  const configPath = path.join(configDir, 'config.json');

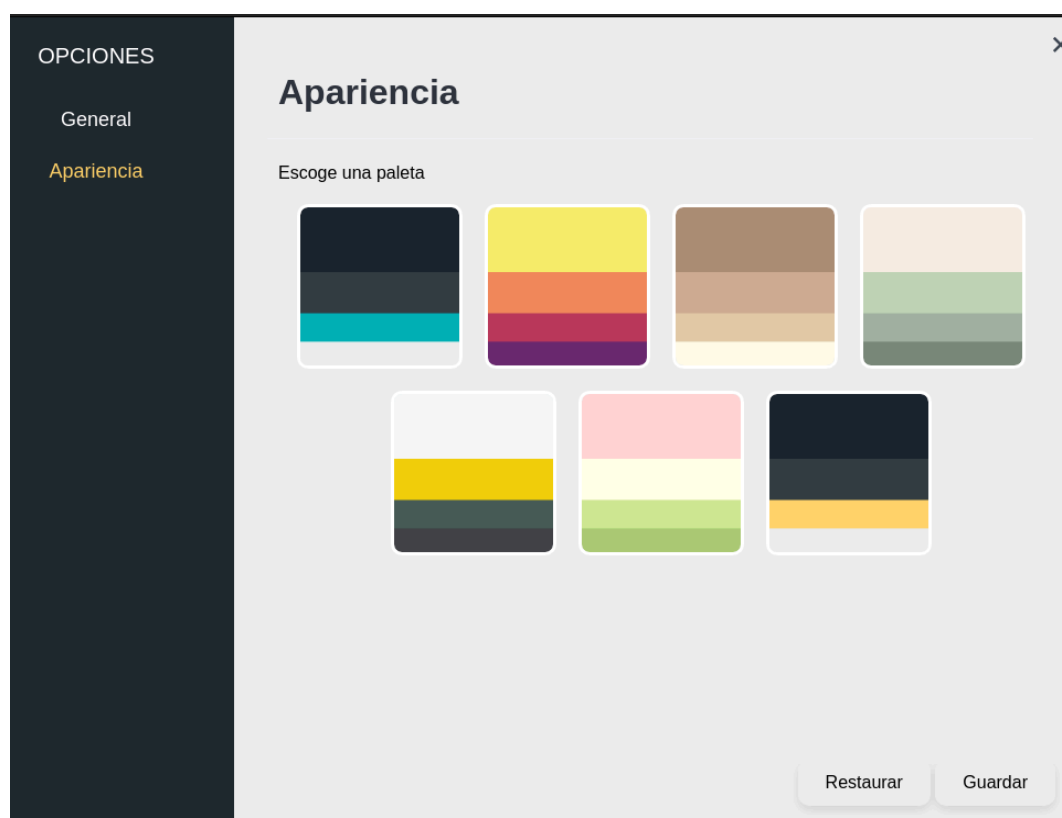
  // Ensure the directory exists
  if (!fs.existsSync(configDir)) {
    fs.mkdirSync(configDir, { recursive: true });
  }
  if (defaultDirectory){
    fs.writeFile(configPath, JSON.stringify(configData, null, 2), (err) => {
      if (err) {
        console.error('Error writing config file', err);
      } else {
        console.log('Config file saved successfully');
        alert("Configuración guardada.");
      }
    });
  }
  else{
    alert("Por favor, selecciona un directorio");
  }
});
```

Apariencia nos permitirá cambiar los colores de la aplicación.



Para ello he decidido crear unas cuantas paletas de colores.

Por ejemplo, la última paleta se parece a la primera, pero cambia el color a amarillo. Solo pondré esta imagen para que podáis probarlo vosotros mismos.



La forma en la que lo he implementado es muy sencilla, variables. Creé un css con solo las modificaciones de colores

```

src
  HTML-CSS
    # calendar.css
    <> calendar.html
    # colours.css
    # index.css
    <> index.html
    # options.css
    <> options.html M
  3 body {
  4   background: var(--primary-bg-colour);
  5 }
  6
  7 /* Sidebar */
  8 .sidebar {
  9   background: var(--secondary-colour);
 10 }
 11
 12 .sidebar li .icon {
 13   color: var(--font-light-colour);

```

Ahí utilizo todas las variables de color de todos los HTML. Después creé un .json con todas las paletas que voy a utilizar. Ahí las separo por ids y pongo las variables.

```

options.html M  {} palettes.json X
src > {} palettes.json > {} 7 > --primary-bg-colour
1  {
2    "1": {
3      "--primary-bg-colour": "#e4e9f7",
4      "--secondary-bg-colour": "#393E46",
5      "--primary-colour": "#00ADB5",
6      "--secondary-colour": "#222831",
7      "--font-dark-colour": "#333333",
8      "--font-light-colour": "#FFFFFF"
9    },
10   "2": {
11     "--primary-bg-colour": "#F9ED69",
12     "--secondary-bg-colour": "#F08A5D",
13     "--primary-colour": "#B83B5E",
14     "--secondary-colour": "#6A2C70",
15     "--font-dark-colour": "#222831",
16     "--font-light-colour": "#FFFFFF"
17   },
18   "3": {
19     "--primary-bg-colour": "#FFFB E9",
20     "--secondary-bg-colour": "#CEAB93",
21     "--primary-colour": "#E3CAA5",
22     "--secondary-colour": "#AD8B73",
23     "--font-dark-colour": "#4A4A4A",
24     "--font-light-colour": "#F0F0F0"
25   },
26   "4": {
27     "--primary-bg-colour": "#F8EDE3",
28     "--secondary-bg-colour": "#BDD2B6"

```



Después cargo el fichero y la guardo en la variable savedColorVariables

```
let savedColorVariables = loadPallete();
```

que se organizan en una hash table (diccionario).

```
function loadPallete() {  
  const configPath = path.join(__dirname, "..", "palettes.json");  
  try {  
    const data = fs.readFileSync(configPath, 'utf8');  
    const config = JSON.parse(data);  
    return config;  
  } catch (err) {  
    console.error('Error reading palettes file:', err);  
  }  
}
```

Cuando le damos a guardar se actualiza el config.json con la id de la paleta seleccionada

```
const saveButton = document.getElementById('appearance-save');  
saveButton.addEventListener('click', () => {  
  const configData = loadConfig();  
  configData.palette = selectedPalette;  
  const configPath = path.join(configDir, 'config.json');  
  
  fs.writeFile(configPath, JSON.stringify(configData, null, 2), (err) => {  
    if (err) {  
      console.error('Error saving palette config file:', err);  
    } else {  
      console.log('Palette config saved successfully');  
      alert("Paleta guardada.");  
    }  
  });  
});
```

Ya por último tocaría cambiar el css de las variables que se haría así:

```
// Function to update CSS variables based on the selected palette  
function updateCSSVariables(paletteId) {  
  const palette = savedColorVariables[paletteId];  
  if (palette) {  
    Object.keys(palette).forEach(variable => {  
      document.documentElement.style.setProperty(variable, palette[variable]);  
    });  
    selectedPalette = paletteId;  
  }  
}
```



Claro que para que funcione en otras páginas habrá que hacer lo mismo de cargar los dos json y actualizar las variables.

d. Resultados

Ya he enseñado la mayoría de la aplicación en la implementación, por tanto aquí quiero que vosotros mismos os descarguéis la aplicación y la probéis.

Debido a la falta de tiempo, no he podido probar que la aplicación funcione en windows, por tanto solo se puede instalar en Linux.

También adjunto el github con dónde podréis ver todos los commits y el código fuente, ya que ese era otro de los objetivos.

Descargar: [Linux](#)

Github: <https://github.com/Lyvmi/Minotaur>

e. Conclusiones

Este ha sido un proyecto complicado, con mucha investigación... por no decir todo. Ha habido un par de veces (2) que he tenido que borrar días de trabajo porque no había enfocado bien el proyecto.

Tampoco me he hecho un favor a mi mismo, ya que no he sido muy responsable y he dejado el proyecto para las últimas semanas.

Sin embargo, estoy orgulloso de mi proyecto, pese a no haber conseguido que funcionen algunas herramientas y haber tenido que dejar una función sin implementar, creo que el resto está muy bien.

He dedicado una gran cantidad de horas a este proyecto y espero que haya quedado reflejado en el resultado.

Tal vez la aplicación contenga algún bug que no haya visto o alguna herramienta funcione mal, pero no tengo tiempo para arreglarlo y muchas veces supone una inversión de tiempo que no tengo.



f. Nuevas propuestas

Como primera propuesta sería terminar las funciones que se han quedado incompletas, principalmente por falta de tiempo.

Tal vez la aplicación contenga algún bug que no haya visto o alguna herramienta funcione mal, pero no tuve tiempo de arreglarlo ya que muchas veces supone una inversión de tiempo que no tenía, por tanto, me gustaría poder arreglarlos.

Para finalizar, me gustaría añadir, que pienso terminar este proyecto más adelante y dejarlo pulido, tal vez añadiendo más herramientas o expandiendo su funcionalidad. Si alguno tiene curiosidad, podéis mirar el [github](#).

Además, doy permiso para que se enseñe en clase a modo de ejemplo si así se quisiera.



5. Anexos

a. Referencias bibliográficas

<https://nodejs.org/api/modules.html>

<https://www.electronjs.org/docs/latest/>

<https://www.electronforge.io/>

<https://chatgpt.com/>

Muchos videos sobre electron que no he apuntado...

Y más que tampoco he apuntado...

e. Código fuente

Al contener más de 1000 líneas de código no voy a adjuntar el código en imágenes.

Enlace a github: <https://github.com/Lyvmi/Minotaur>