Universidad Nacional de Ingeniería

Recinto Universitario Pedro Arauz Palacios

Facultad de Ciencias y Sistemas

Proyecto de Diseño de Sistemas de Internet

Integrantes:

- **Freddy Andrés López Ramírez**
- **Lenox Shaheim Wiltshire Espinoza**
- **Jairo Ramón Zamora Ortega**

Docente: Lázaro González

Martes 22 de Noviembre, 2022

Descripción:

El presente proyecto es un bloc de notas el cual funciona como una colección de notas, en las cuales se pueden adjuntar otras notas secundarias. El presente proyecto hace uso de la arquitectura REST, con el propósito de realizar cambios con la base de datos por medio de un servidor.

1. Las tecnologías utilizadas para la elaboración de este son las siguientes:

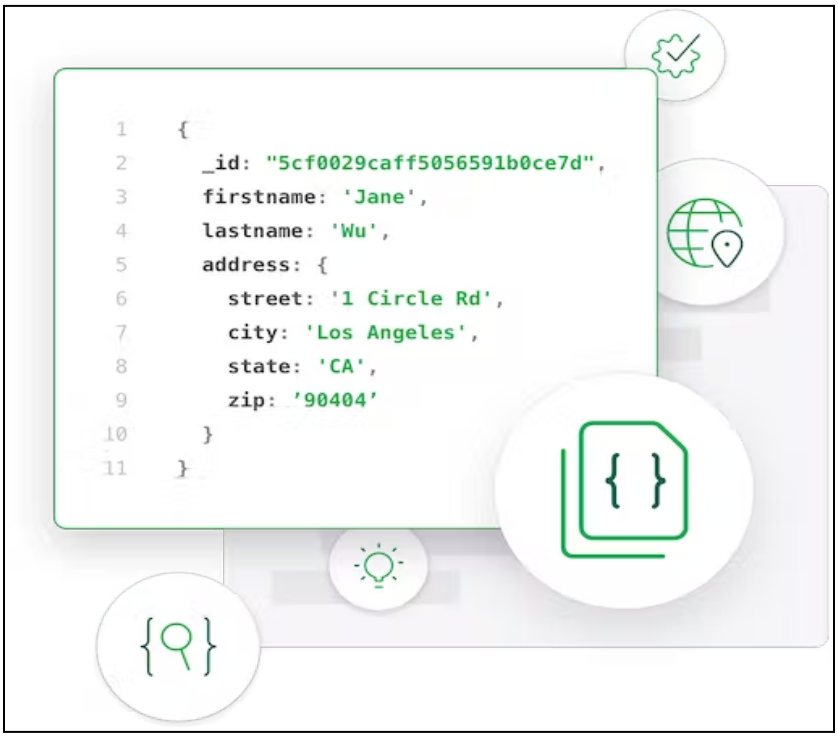
- **MongoDB = Base de datos:**

  - MongoDB almacena datos en documentos flexibles similares a JSON, por lo que los campos pueden variar entre documentos y la estructura de datos puede cambiarse con el tiempo.
  - El modelo de documento se asigna a los objetos en el código de su aplicación para facilitar el trabajo con los datos.
  - Las consultas ad hoc, la indexación y la agregación en tiempo real ofrecen maneras potentes de acceder a los datos y analizarlos.
  - MongoDB es una base de datos distribuida en su núcleo, por lo que la alta disponibilidad, la escalabilidad horizontal y la distribución geográfica están integradas y son fáciles de usar.

- **Express = Conexion**

  - Express es un framework de backend Node.js minimalista, rápido y similar a Sinatra, que proporciona características y herramientas robustas para desarrollar aplicaciones de backend escalables. Te ofrece el sistema de enrutamiento y características simplificadas para ampliar el framework con componentes y partes más potentes en función de los casos de uso de tu aplicación.

- El framework proporciona un conjunto de herramientas para aplicaciones web, peticiones y respuestas HTTP, enrutamiento y middleware para construir y desplegar aplicaciones a gran escala y preparadas para la empresa.



- **React = Interface**

- ReactJS es una de las librerías más populares de JavaScript para el desarrollo de aplicaciones móviles y web. Creada por Facebook, React contiene una colección de fragmentos de código JavaScript reutilizables utilizados para la construcción de la interfaz de usuario (UI) llamados componentes.

- **Node = Servidor**

- Node.js es un entorno *controlado por eventos* diseñado para crear aplicaciones escalables, permitiéndote establecer y gestionar múltiples conexiones al mismo tiempo. Gracias a esta característica, no tienes que preocuparte con el bloqueo de procesos, pues no hay bloqueos.

2. ***Una vez elegidas las tecnologías para la elaboración del proyecto, se procedió a elegir una página en donde se hará el deployment del proyecto:***

En este proceso se eligió Hostinger para alojar el proyecto debido a que este proporciona de un servicio de VPS, el cual es un servicio en donde se le proporciona al usuario de un entorno virtual en donde se retenga su información, lo cual evita que los usuarios compartan recursos entre ellos.

El dominio elegido para el proyecto el siguiente: https://flashcardsage.com/

Una vez hecho esto, se procedió tanto a instalar las dependencias como a instalar las librerías que se utilizaran para la elaboración del proyecto por medio del comando **npm install && npm run dev.** Todas las dependencias se pueden encontrar en el paquete **JSON** pero entre las más importantes tenemos **Express**, **Mongoose** y **Nodemon**.

Express ha como se había mencionado previamente, tiene como funcionalidad hacer uso de los diferentes métodos para realizar consultas entre **el frontend** con **el backend**. **Mongoose** es utilizado para hacer uso de los diferentes métodos para proporcionar la estructura de los esquemas de la base de datos, las cuales serán enviadas por medio de **Express** a **MongoDB** y por último, **Nodemon** es utilizado para no tener que reiniciar el servidor cada vez que se realiza un cambio en el código.

El código fue escrito mayormente en **Typescript** y **React** debido a que Typescript proporciona una mayor información sobre las diferentes clases que se pueden utilizar para el desarrollo de **páginas web** y React nos permite escribir código de como se verá la página en **Typescript** donde a pesar de esto, será procesada como que si fue escrita en **HTML**. Esto permite que al momento de realizar una inspección de la página por parte del cliente, el código se mantendrá oculto. Apartando esto, debido a que se hace necesario conectarse con la base de datos, se crea un usuario y una contraseña, la cual fue oculta por medio de un **.env file**, el cual no fue subido al repositorio debido a que fue ignorado por el .**gitignore**.

3. ***Se procedió a crear dos folders, uno para el lado del cliente, y otro, para el lado del servidor:***

Se empezó con el lado del servidor en donde se instalaron las instancias mencionadas previamente mencionadas las cuales están referenciadas en el **package.json**. En el **folder source**, se crearon los folders models y controllers con el fin de seguir el patrón de diseño **MVC** que separa el programa entre **Modelo**, **Vista** y **Controlador**. Donde el modelo, funciona como una colección de clases en donde se describe la estructura de las clases con las cuales se trabajara. Las vistas son aquellas ventanas donde se verán los esquemas y por último, los controladores son todas aquellas funciones que se utilizan para relacionar el cliente con el servidor.

En **deck** se creó la estructura de las notas mientras que en **controllers**, se crearon los métodos de **crear**, **borrar** y **obtener** información desde el frontend hasta la base de datos. Esto se realiza haciendo uso de los métodos de la arquitectura **restful API** previamente mencionada.

➢ **Models:**

13 lines (9 sloc) | 266 Bytes — Raw | Blame

```
1  import mongoose from "mongoose";
2
3  const Schema = mongoose.Schema;
4  // const ObjectId = mongoose.Types.ObjectId;
5
6  const DeckSchema = new Schema({
7    title: String,
8    cards: [String],
9  });
10
11 const DeckModel = mongoose.model("Deck", DeckSchema);
12
13 export default DeckModel;
```

➢ **Controllers:**

12 lines (11 sloc) | 408 Bytes — Raw | Blame

```
1  import { Request, Response } from "express";
2  import Deck from "../models/Deck";
3
4  export async function createCardForDeckController(req: Request, res: Response) {
5    const deckId = req.params.deckId;
6    const deck = await Deck.findById(deckId);
7    if (!deck) return res.status(400).send("no deck of this id exists");
8    const { text } = req.body;
9    deck.cards.push(text);
10   await deck.save();
11   res.json(deck);
12 }
```

Después de esto, se procedió a trabajar la parte del cliente, siendo esta mayormente el frontend y como se conecta esta al lado del servidor. Nuestra página principal es **main.tsx** en donde se importan las clases en donde se describe la estructura, apariencia y funcionalidad de las páginas. A su vez, se creó un apartado de **API**, en donde, como se había dicho previamente, se escribieron las funciones de la arquitectura **restful API**, las cuales serán utilizadas para conectarse a la base de datos.

➢ Header:

21 lines (18 sloc) | 354 Bytes — Raw | Blame

```
import "./Header.css";

export function Header() {
  return (
    <div className="Header">
      <div className="container">
        <div>
          <a href="/">FlashCardSage</a>
        </div>

        <div>
          <a href="/">Decks</a>
        </div>

        <div>
          <a href="/login">login</a>
        </div>
      </div>
    </div>
  );
}
```

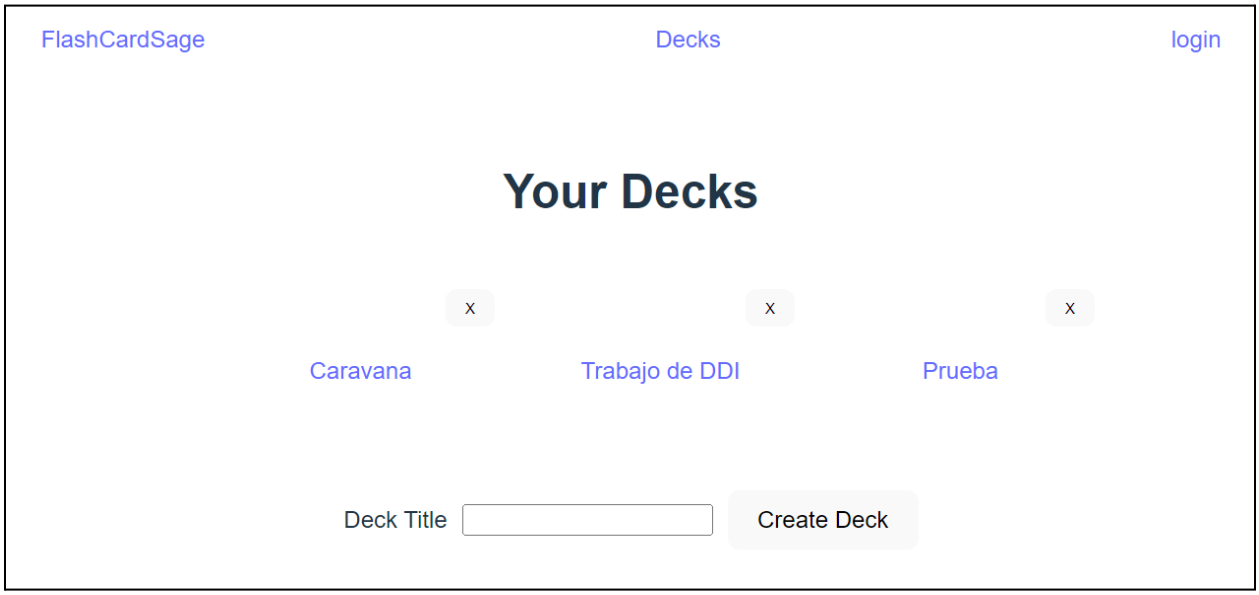
➢ API:

14 lines (13 sloc) | 306 Bytes — Raw | Blame

```
import { API_URL } from "./config";

export async function createDeck(title: string) {
  const response = await fetch(`${API_URL}/decks`, {
    method: "POST",
    body: JSON.stringify({
      title,
    }),
    headers: {
      "Content-Type": "application/json",
    },
  });
  return response.json();
}
```

Habiendo realizado esto, tenemos el producto final, el cual está alojado en el **main.ts**, que es nuestra página principal en donde se importan todas las clases trabajadoras. El producto final terminó siendo el siguiente:



Esta página puede ser utilizada como un **bloc de notas** entre miembros de un grupo. Un ejemplo de esto, puede ser una familia para corroborar que dejaron la puerta enllavada. Entre las bondades del programa, tenemos que es bastante intuitivo, lo cual resulta ser algo muy importante al momento de realizar un programa orientado donde hayan personas mayores. Entre las **debilidades**, tendríamos la **interfaz** muy poca trabajada, la cual puede mejorarse por medio de la utilización de **bootstrap** o **tailwind css**. Apartando esto, hay un problema con la resolución de excepciones en la autenticación de usuario cuando no se tiene a ningún usuario. Esto se pudiera mejorar creando un **exception handler** al momento de tener ese error.