CENTRO UNIVERSITÁRIO PRESBITERIANO MACKENZIE

FACULDADE DE COMPUTAÇÃO E INFORMÁTICA CURSO

DE SISTEMAS DE INFORMAÇÃO

ESTEPHANY SILVA

PEDRO CAIXETA

VICTOR GARGALHONE

PROJETO EXODIA

ESTEPHANY SILVA

PEDRO CAIXETA

VICTOR GARGALHONE

PROJETO EXODIA

Relatório apresentado à disciplina de Web Mobile do curso de Sistemas de Informação do Centro Universitário Presbiteriano Mackenzie, como parte dos requisitos para avaliação do Projeto 2.

Orientador: Professora Ma. Paula Leite

SUMÁRIO

1. INTRODUÇÃO

O presente documento tem como finalidade apresentar a proposta inicial do Projeto Exodia, a ser desenvolvido na disciplina de Web Mobile. O projeto tem como foco o uso prático do framework NextJS em conjunto com o consumo de APIs externas, promovendo o aprendizado aplicado dos conteúdos ministrados em aula.

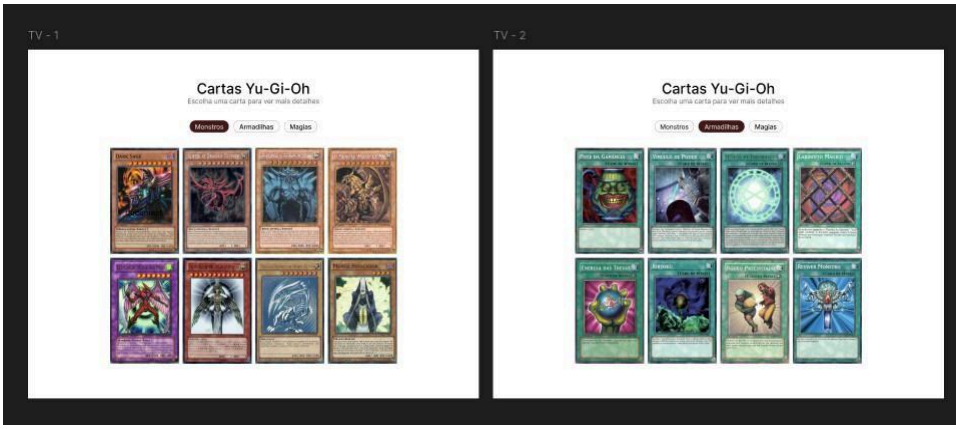
2. OBJETIVO

O objetivo do Projeto 2 é consolidar nosso conhecimento sobre a utilização do NextJS por meio do desenvolvimento de um site que permita a busca de cartas do universo Yu-Gi-Oh! utilizando uma API pública.

3. PRODUTO FINAL

O produto final será um site funcional desenvolvido com NextJS, capaz de realizar consultas a uma API pública de cartas de Yu-Gi-Oh! e exibir as informações de forma clara, organizada e responsiva

Protótipo de Telas



Link do figma: https://www.figma.com/design/GcbQE5t1UNMRGsZSBaPSEh/Untitled?node-id=0-1&t=Gi0OuijYxmGIq771-1

Link do Site: https://projeto-yugioh-nextjs-demo.vercel.app/

4. METODOLOGIA

O projeto será conduzido com uma abordagem prática e interativa, respeitando as seguintes etapas:
Estruturação do projeto com NextJS;

Implementação da interface de usuário com foco em usabilidade;

Integração com a API pública de Yu-Gi-Oh!;

Testes e melhorias contínuas.

O grupo adotará práticas de versionamento com Git e usará ferramentas como VS Code e Node.js para desenvolvimento.

5. CRONOGRAMA

Parte 1 - Ideação e estrutura inicial: 30/04/2025 Parte 2 - Projeto funcional sem API: 21/05/2025
Apresentação Final - Projeto completo com API integrada: 28/05/2025

6. RECURSOS NECESSÁRIOS

Os seguintes recursos serão utilizados durante o desenvolvimento:

- Computadores com ambiente de desenvolvimento configurado;
- Framework NextJS;
- Acesso à API pública de Yu-Gi-Oh!;
- Ferramentas de desenvolvimento como Visual Studio Code, Git e navegadores atualizados.

O grupo já possui acesso a todos os recursos necessários.

7. AVALIAÇÃO E RESULTADOS ESPERADOS

A avaliação do projeto será feita com base em critérios como:

- Cumprimento dos prazos;
- Estrutura adequada do código;
- Funcionalidade da aplicação;
- Consumo correto da API;
- Clareza e organização da interface;
- Apresentação final do projeto.

8. DESCRIÇÃO DA ESTILIZAÇÃO EM CSS

8.1 Introdução

O presente tópico detalha a parte de estilização em CSS do Projeto Exodia. O design foi cuidadosamente baseado na estética tradicional do anime Yu-Gi-Oh!, respeitando suas cores, fontes e estilos gráficos característicos. O objetivo foi criar uma interface coerente com o universo clássico da obra, promovendo uma experiência visual imersiva.

8.2 Processo de Ideação

A concepção do design teve como princípio transmitir a atmosfera mística e competitiva de Yu-Gi-Oh!, utilizando elementos visuais que remetessem ao anime e ao jogo de cartas original. As cores predominantes escolhidas foram roxo, dourado, preto e vermelho, com tipografia estilizada. As principais referências foram os menus dos jogos da franquia e as bordas tradicionais das cartas.

8.3 Tipo de Website Escolhido

O website foi estruturado como um catálogo de cartas, com filtros e organização em grid, proporcionando uma navegação prática e visual, similar a um acervo ou ao tabuleiro de jogo tradicional.

8.4 Desenvolvimento e Explicação do Código CSS

A estilização foi organizada em quatro arquivos principais:

- `globals.css`
- `pagemodule.css`
- `Headermodule.css`
- `Footermodule.css`

8.4.1 Arquivo globals.css

Definimos as variáveis CSS para possibilitar a alternância entre o modo claro e o modo escuro, proporcionando uma experiência visual personalizada.

```
:root {
  --background: #fffffcb;
  --foreground: #00014d;
  --link-color: #8400ff;
}
```

```
body.dark-mode {
  --background: #190027;
  --foreground: #ffee00;
  --link-color: #ff0000;
}
```

A fonte `'Zen Dots'` foi escolhida para reforçar o aspecto tecnológico e enigmático do universo do anime.

8.4.2 Arquivo pagemodule.css

Foram definidas regras para hierarquia visual, com títulos de tamanhos distintos:

```
.title {
  font-size: 2.5rem;
  font-weight: bold;
}
```

Os botões de filtro foram estilizados com bordas arredondadas e cores fortes:

```
.filter {
  background-color: #5a2d2d;
  color: white;
  border-radius: 999px;
}
```

Os cards foram organizados em grid para simular o tabuleiro de cartas:

```
.cardGrid {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
}
```

A responsividade foi garantida com media queries, assegurando boa visualização em diversos dispositivos.

8.4.3 Arquivo Headermodule.css

O cabeçalho apresenta fundo preto e borda branca, evocando as bordas tradicionais das cartas. A logo é interativa e responsiva:

```
.logoImage {
  max-height: 60px;
  cursor: pointer;
}
```

O botão de alternância de tema tem cor personalizada :

```
.themeButton {
  background-color: #0070f3;
}
```

Os ícones ganham destaque com efeitos `hover`:

```
.icon:hover {
  transform: scale(1.2);
}
```

8.4.4 Arquivo Footermodule.css

O rodapé possui estilização simples e funcional, com fundo preto e texto branco:

```
.footer {
  background-color: #000;
  color: white;
}
```

8.4.5 Arquivo CardModule.css

Os cards foram estilizados para simular profundidade e tridimensionalidade:

```
.card {
  border-radius: 8px;
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.2);
}
```

8.5 Responsividade

A responsividade foi assegurada com media queries, adaptando fontes, espaçamentos e layouts conforme a largura do dispositivo, proporcionando acessibilidade e boa experiência de usuário.

8.6 Considerações Finais CSS

A estilização do Projeto Exodia proporcionou a aplicação de importantes conceitos de desenvolvimento web, como:

- Modularização e Responsividade do CSS;
- Uso de variáveis para gestão de temas;
- Fidelidade à identidade visual do universo Yu-Gi-Oh!.

9 Desenvolvimento e explicação do código Next.js

A aplicação do Projeto Exodia é construída a partir de componentes Next.js que operam em conjunto para gerenciar a interface e a lógica funcional. A estrutura é modular, com arquivos `.js` dedicados a funcionalidades específicas, otimizando a organização do projeto.

9.1 Arquivo layout.js

Este arquivo é responsável por definir a estrutura global da aplicação, incluindo metadados e componentes fixos.

Importação de fontes: As fontes Geist e Geist_Mono são importadas do `next/font/google` para utilização no projeto. Adicionalmente, a fonte 'Zen Dots', do Google Fonts, é incorporada via link no cabeçalho HTML, reforçando a estética tecnológica e enigmática do universo do anime.

```
import { Geist, Geist_Mono } from "next/font/google";

// ...

export default function RootLayout({ children }) {

  return (

    <html lang="en">

      <head>

        {/* Google Fonts Zen Dots */}

        <link rel="preconnect"
href="https://fonts.googleapis.com" />
```

```
        <link rel="preconnect" href="https://fonts.gstatic.com"
crossOrigin="" />

        <link
href="https://fonts.googleapis.com/css2?family=Zen+Dots&display=
swap"

              rel="stylesheet"/>

      </head>

      {/* ... */}

    </html>

  );

}
```

Adição de componentes fixos: Os componentes `Header` e `Footer` são importados e renderizados, assegurando sua presença em todas as páginas da aplicação.

Definição de metadados: A constante `metadata` estabelece o título ("Cartas Yu-Gi-Oh") e a descrição ("Grupo Pichu") da página, informações cruciais para navegadores e otimização de busca (SEO).

```
export const metadata = {

title: "Cartas Yu-Gi-Oh",

  description: "Grupo Pichu",

};
```

Estrutura HTML base: O `RootLayout` constrói o esqueleto HTML fundamental da página, aplicando as fontes e integrando os componentes de cabeçalho e rodapé.

```
export default function RootLayout({ children }) {

  return (
```

```
      <html lang="en">
        {/* ... */}
        <body>
          <Header />
          <main>{children}</main>
          <Footer />
        </body>
      </html>
    );
  }
```

9.2 Arquivo page.js

Este é o componente da página inicial da aplicação, responsável por buscar e exibir as cartas de Yu-Gi-Oh!.

Gerenciamento de dados e estado: Utiliza `useState` para controlar a lista de cartas, o status de carregamento e o registro de possíveis erros.

```
export default function Home() {
  const [cards, setCards] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  // ...
}
```

Busca de cartas da API: O hook `useEffect` executa uma requisição à API de Yu-Gi-Oh! (https://db.ygoprodeck.com/api/v7/cardinfo.php?type=Normal%20Monster) quando a página é

carregada.

```
useEffect(() => {

fetch("https://db.ygoprodeck.com/api/v7/cardinfo.php?type=Normal
Monster")
    .then((res) => { /* ... */ })
    .then((data) => setCards(data.data.slice(0, 8))) // Limita a
8 cartas
    .catch((err) => setError(err.message))
    .finally(() => setLoading(false));
}, []);
```

Exibição de status: Mensagens como "Carregando cartas..." ou indicativos de erro são exibidas de acordo com o status da requisição.

```
if (loading) return <p>Carregando cartas...</p>;
if (error) return <p>Erro: {error}</p>;
```

Renderização da interface: A página apresenta um título, um botão de filtro e uma grade onde as cartas são exibidas, utilizando o componente `Card` para cada item.

```
return (
  <main className={styles.container}>
    <h1 className={styles.title}>Cartas Yu-Gi-Oh</h1>
    <p className={styles.subtitle}>Escolha uma carta para ver
mais detalhes</p>
    <nav className={styles.buttons}>
      <button className={styles.filter}>Monstros</button>
    </nav>
    <section className={styles.cardGrid}>
      {cards.map((card) => (
        <Card
```

```
            key={card.id}
            image={card.card_images[0]?.image_url}
            name={card.name}
          />
        ))}
      </section>
    </main>
  );
```

9.3 Arquivo Card.js

Este componente representa uma única carta interativa na interface da aplicação.

Componente de carta: Recebe as propriedades `image` (URL da imagem da carta) e `name` (nome da carta).

Link para detalhes: A carta é renderizada como um link, direcionando o usuário para uma página de detalhes específica, utilizando o nome da carta codificado na URL.

```
'use client'
import Link from 'next/link'
import styles from './Card.module.css'

export default function Card({ image, name }) {
  const cardName = encodeURIComponent(name)
  return (
    <Link href={`/${cardName}`} className={styles.card}>
      <img src={image} alt={name} />
    </Link>
  )
}
```

9.4 Arquivo Footer.js

Este componente define o rodapé da aplicação.

Estrutura simples: Cria um elemento `<footer>` com os estilos definidos para o rodapé da página.

```
'use client';
import styles from './Footer.module.css';

const Footer = () => {
  return (
    <footer className={styles.footer}>
      <p> {}</p> {/* Conteúdo do rodapé */}
    </footer>
  );
};
export default Footer;
```

9.5 Arquivo Header.js

Este componente define o cabeçalho da aplicação, incluindo o logo e a funcionalidade de alternância de tema.

Gerenciamento de tema: Utiliza `useState` para controlar o estado do modo claro/escuro e `useEffect` para carregar a preferência de tema salva pelo usuário no `localStorage` ou do sistema.

```
'use client';
import { useEffect, useState } from 'react';
// ...
const Header = () => {
```

```
  const [isDarkMode, setIsDarkMode] = useState(false);
  useEffect(() => {
    const savedMode = localStorage.getItem('theme');
    if (savedMode) {
      setIsDarkMode(savedMode === 'dark');
    } else {
      setIsDarkMode(window.matchMedia &&
window.matchMedia('(prefers-color-scheme: dark)').matches);
    }
  }, []);
  // ...
};
```

Alternância de tema: A função `toggleTheme` inverte o estado do tema, salva a nova preferência no `localStorage` e aplica/remove a classe `dark-mode` ao `body` do documento para refletir as mudanças visuais.

```
const toggleTheme = () => {
  setIsDarkMode(!isDarkMode);
  const newTheme = !isDarkMode ? 'dark' : 'light';
  localStorage.setItem('theme', newTheme);
  document.body.classList.toggle('dark-mode', newTheme ===
'dark');
};
```

Logo e botão de tema: O cabeçalho renderiza o logo como um link clicável que direciona para a página inicial, além de um botão interativo para alternar entre os modos de tema claro e escuro.

```
return (
  <header className={styles.header}>
    <Link href="/" className={styles.logo}>
      <img src="/images/logo-main.png" alt="Yu-Gi-Oh! Logo"
className={styles.logoImage} />
    </Link>
    <button className={styles.themeButton}
onClick={toggleTheme}>
      <img
      src={isDarkMode ? '/icons/sun-svgrepo-com.svg' :
'/icons/moon-svgrepo-com.svg'}
      alt={isDarkMode ? 'Modo Claro' : 'Modo Escuro'}
          className={styles.icon}
        />
      </button>
  </header>
);
```

9.6 Considerações finais Next.js

- Aprendemos a usar Rota dinâmica usando App Router;
- Usamos o UseState para preencher componentes (As cartas);
- Consumimos uma API com Fetch e UseEffect;
- Componentização de funções.