# Documentazione

## Sommario



## Un’introduzione a MukReact

Seguo il mondo dei Pokémon da quando sono bambino, e nelle mie squadre è spesso stato presente un pokemon in particolare : Muk. MukReact è, quindi, una applicazione web creata con lo scopo di onorare uno dei miei mostri preferiti, specialmente considerando che tutto sommato è un pokemon relativamente impopolare.

L’applicazione è stata sviluppata con la libreria React, come parte della valutazione per il corso di “Tecnologie e Applicazioni dei Sistemi Distribuiti” (2023/2024) del corso di laurea magistrale “Teoria e Tecnologia della Comunicazione” dell'Università degli Studi di Milano-Bicocca.

## 1 La struttura generale dell’app

L’applicazione web presenta tre pagine principali:

- “Home”, pagina introduttiva del sito e che ne presenta brevemente i contenuti;

- “Appearances”, sezione che si occupa di mostrare le varie apparizioni di Muk nel gioco di carte collezionabili; questa pagina può essere visualizzata sia come insieme di carte che come tabella, e permette al click sulla carta di vederne dei dettagli. Quest’ultimi vengono gestiti da un altro componente “MukDetails”;
- “Info”, pagina dove sono presenti informazioni sul Muk come Pokemon.

Il front-end del progetto è stato diviso in diverse parti categorizzate all’interno della cartella “src” in questo modo:

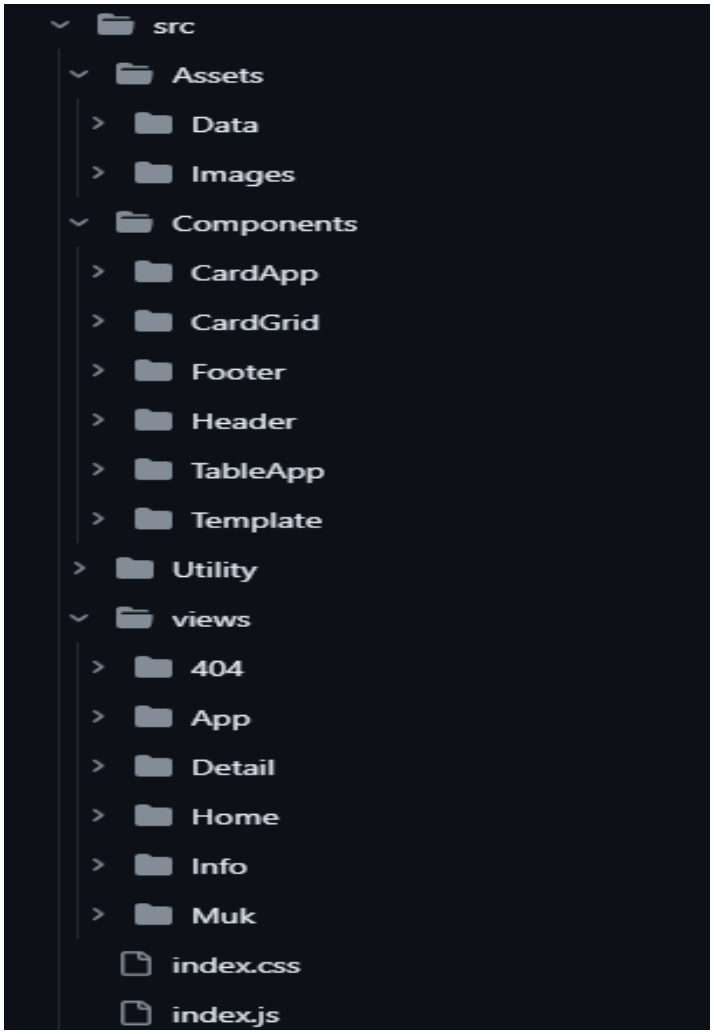
- “Assets”, che contiene file statici e ulteriormente suddivisa in:
    - “Images”, dove sono presenti tutte le immagini usate nel sito;
    - “Data”, dove è presente un file JSON, con delle caratteristiche delle carte( id, set, immagini…) utile al popolamento della pagina “Appearances”.
- “Components”, che contiene i vari componenti inseriti nella web app e si divide in:
    - “Header”;
    - “Footer”;
    - “Template”;
    - “CardGrid” e “CardApp”;
    - “TableApp”.

- “Views”, che contiene le pagini navigabili del sito. Oltre alle tre citate precedentemente troviamo:
    - “App”, pagina di base importata in *index.js,* utilizzata per la definizione del routing e per definire i props per il template;
    - “Error”, pagina utilizzata per gestire l’errore 404.
- “Utility”, che contiene delle funzioni utili alla coerenza e pulizia visiva del sito.

Ogni pagina è stata fornita con un “CSS.module” per definirne l’estetica, in aggiunta al oglio di stile “index.ccs” dove sono stati dichiarati stili globali. Sono state inoltre utilizzate librerie esterne e API come:

- Reactstrap, che importa il toolkit di Bootstrap in React, e utilizzato a supporto dei vari moduli css per l’abbellimento del sito;
- Il pacchetto clsx, utilizzato per gestire lo stile del pulsante di switch per il layout griglia-tabella in “Appearances”.

- React-router-dom, poiché React non dispone di un sistema di routing integrato, e utilizzata proprio per permettere l'effettiva navigazione tra le pagine del sito.
- TGCdex, per ottenere i dati con cui popolare la pagina "Details" delle carte;
- PokeAPI, per ottenere i dati di Muk con cui popolare la pagina "Info".




# 2 Componenti

## 2.1 Template

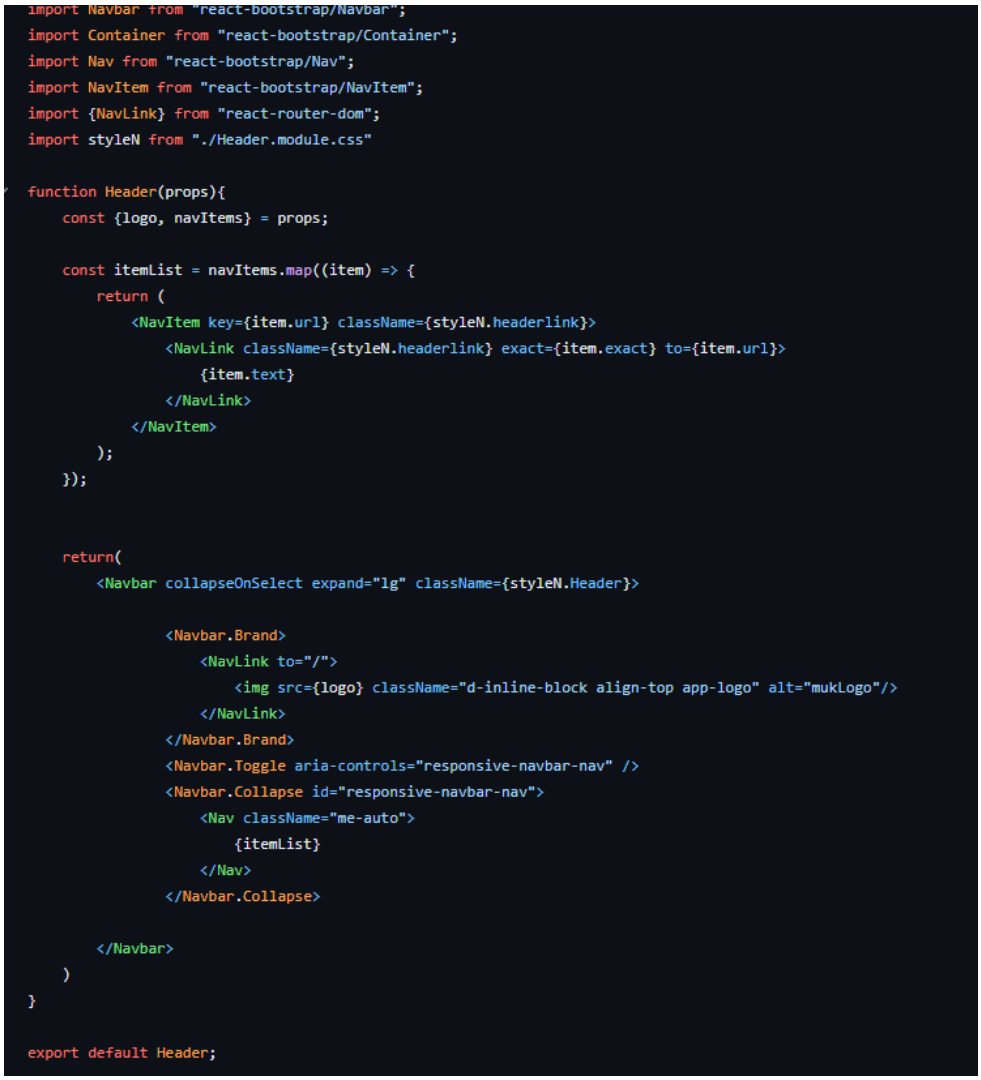
Il componente stateless "Template" viene utilizzato per il rendering della struttura del sito. Contiene a sua volta i componenti "Footer" e "Header, a cui passa dei props a sua volta ricevuti da "App.js" ; tra questi due componenti è inserito l'oggetto *{children},* utilizzato per il rendering del contenuto della pagina.

```jsx
import React from "react";
import Footer from "../Footer/Footer";
import Header from "../Header/Header";
import styleT from "./template.module.css"

function Template(props){
    const {footerCourseName, footerCourseLink, logo, navItems, children} = props;
    return(
        <div className={styleT.templateApp}>
            <Header
                logo={logo}
                navItems={navItems}/>
            <main>
                {children}
            </main>
            <Footer
                courseName = {footerCourseName}
                courseLink = {footerCourseLink}
                logo={logo}
                navItems={navItems}/>
        </div>
    )
}

export default Template;
```

## 2.2 Header

Il componente stateless “Header” ,come dice il suo nome, visualizza l’header del sito.

Nel componente viene creata un  oggetto costante, in cui viene “mappata” (.map) la prop array “navitems” , e restituito ogni key  dell’array come componente della NavBar, che vengono racchiusi in un tag “NavLink” per permettere il routing; la costante verrà poi inserità all’interno del rendering del componente. La struttura dell’header, dopo il return, è stata costruita utilizzando le funzionalità di Bootstrap.

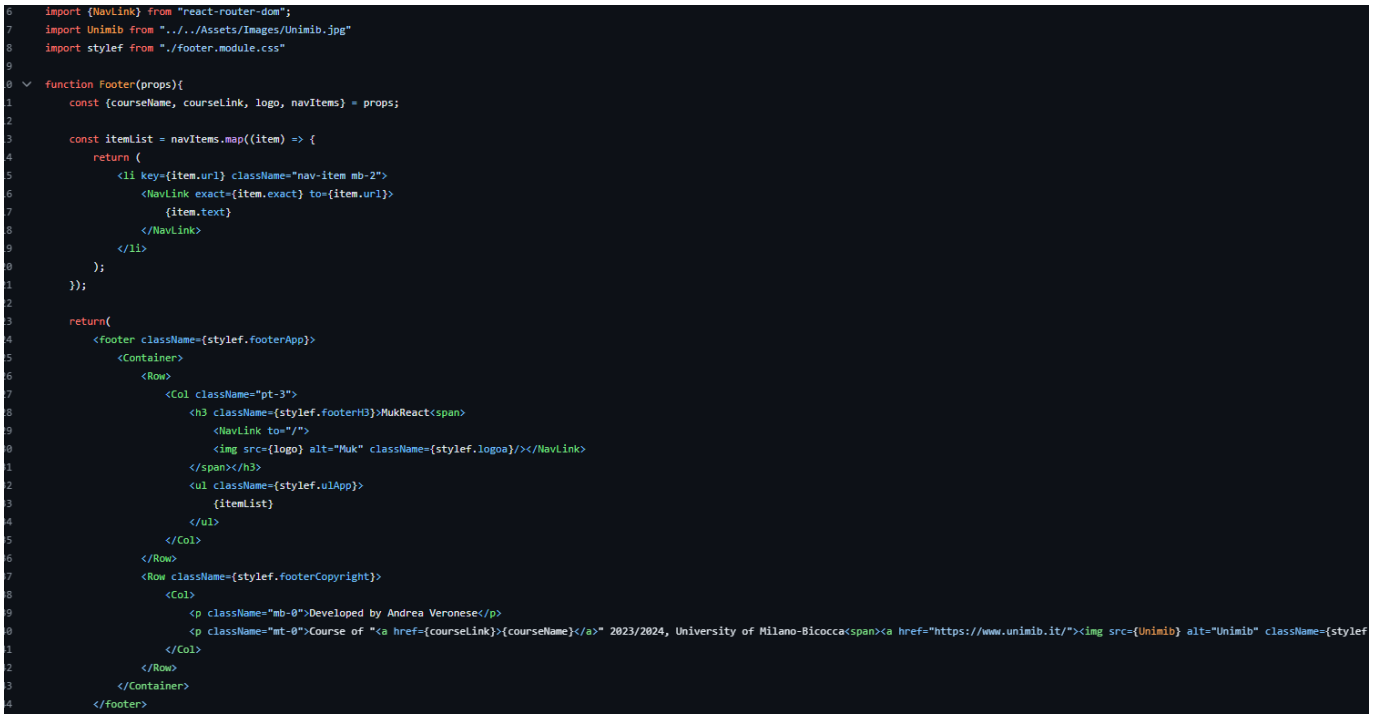


```
import Navbar from "react-bootstrap/Navbar";
import Container from "react-bootstrap/Container";
import Nav from "react-bootstrap/Nav";
import NavItem from "react-bootstrap/NavItem";
import {NavLink} from "react-router-dom";
import styleN from "./Header.module.css"

function Header(props){
    const {logo, navItems} = props;

    const itemList = navItems.map((item) => {
        return (
            <NavItem key={item.url} className={styleN.headerlink}>
                <NavLink className={styleN.headerlink} exact={item.exact} to={item.url}>
                    {item.text}
                </NavLink>
            </NavItem>
        );
    });


    return(
        <Navbar collapseOnSelect expand="lg" className={styleN.Header}>

                <Navbar.Brand>
                    <NavLink to="/">
                        <img src={logo} className="d-inline-block align-top app-logo" alt="mukLogo"/>
                    </NavLink>
                </Navbar.Brand>
                <Navbar.Toggle aria-controls="responsive-navbar-nav" />
                <Navbar.Collapse id="responsive-navbar-nav">
                    <Nav className="me-auto">
                        {itemList}
                    </Nav>
                </Navbar.Collapse>

        </Navbar>
    )
}

export default Header;
```

## 2.3 Footer

Il componente stateless “Footer” , ovviamente, visualizza il footer del sito.

Questo componente funziona in modo analogo all’header, con la creazione di una costante per la visualizzazione degli elementi di navigazione.



```
import {NavLink} from "react-router-dom";
import Unimib from "../../Assets/Images/Unimib.jpg"
import stylef from "./footer.module.css"

function Footer(props){
    const {courseName, courseLink, logo, navItems} = props;

    const itemList = navItems.map((item) => {
        return (
            <li key={item.url} className="nav-item mb-2">
                <NavLink exact={item.exact} to={item.url}>
                    {item.text}
                </NavLink>
            </li>
        );
    });

    return(
        <footer className={stylef.footerApp}>
            <Container>
                <Row>
                    <Col className="pt-3">
                        <h3 className={stylef.footerH3}>MukReact<span>
                            <NavLink to="/">
                            <img src={logo} alt="Muk" className={stylef.logoa}/></NavLink>
                        </span></h3>
                        <ul className={stylef.ulApp}>
                            {itemList}
                        </ul>
                    </Col>
                </Row>
                <Row className={stylef.footerCopyright}>
                    <Col>
                        <p className="mb-0">Developed by Andrea Veronese</p>
                        <p className="mt-0">Course of "<a href={courseLink}>{courseName}</a>" 2023/2024, University of Milano-Bicocca<span><a href="https://www.unimib.it/"><img src={Unimib} alt="Unimib" className={stylef
                    </Col>
                </Row>
            </Container>
        </footer>
```

## 2.4 CardGrid

CardGrid è un componente stateless che serve a definire la struttura della visualizzazione a carte. Riceve dei dati JSON da visualizzare, via “Appearances”, come prop, li mappa all’interno di una costante e li inoltra a sua volta al componente CardApp. La costante viene poi richiamata e visualizzata a griglia, sfruttando Boostrap per assegnare dei breakpoint a partire dai valori della prop col.

```jsx
import 'bootstrap/dist/css/bootstrap.min.css';
import CardApp from "../CardApp/CardApp";
function CardGrid(props){
    const {mukData, col}=props;
    const mukGrid = mukData.map((muk) => {
        return(
            <div key={muk.id} className="col">
                <CardApp
                    id={muk.id}
                    name={muk.name}
                    image={muk.image}
                    numb={muk.numb}
                    set={muk.set}
                    series={muk.series}
                    date={muk.releaseDate}
                    logo={muk.logo}/>
            </div>
        );
    });

    return(
        <div className={`row
                        row-cols-${col.xs}
                        row-cols-sm-${col.sm}
                        row-cols-md-${col.md}
                        row-cols-lg-${col.lg}
                        row-cols-xl-${col.xl}`
        }>
            {mukGrid}
        </div>
    );
}

export default CardGrid
```

## 2.5 CardApp

Il componente stateless “CardApp” serve per definire la visualizzazione a didascalia.

La struttura è abbastanza semplice, e consiste nella creazione delle effettive “cards” grazie alle informazioni ricevute da “CardGrid”, poi che andranno a popolare quest’ultima;

É possibile cliccare sull’immagine della carta per accederne ai dettagli.

```jsx
import Card from "react-bootstrap/Card";
import CardImg from "react-bootstrap/CardImg";
import Row from "react-bootstrap/Row";
import Col from "react-bootstrap/Col";
import {CardText} from "reactstrap";
import {NavLink} from "react-router-dom";
import mukDefaultImg from "../../Utility/Utility";
import styleCa from "./CardApp.module.css"

function CardApp(props){
    const {id, name, image, numb, set, series, date, logo} = props;

    return(

            <Card className={styleCa.mukCard}>
                <Row className="no-gutters">
                    <Col md={6}>
                        <NavLink to={`/Appearances/${numb}`}>
                            <CardImg onError={(event) => mukDefaultImg(event)}
                                     src={image} alt={name}/>
                        </NavLink>
                    </Col>
                    <Col md={6} >
                        <CardText><b>Series:</b> <span>{series}</span></CardText>
                        <CardText><b>Set: </b> {set}</CardText>
                        <CardText><b>Release date: </b><span>{date}</span></CardText>
                    </Col>
                </Row>
            </Card>
    )
}

export default CardApp
```

## 2.6 Table

Il componente stateless “Table” serve per definire la struttura a tabella delle carte.

Ricevuto i dati come prop, questi vengono mappati e inseriti in una tabella che presenta l’immagine, la serie e il set di appartenenza e la data di uscita.

Come per “CardApp”, è possibile cliccare sull’immagine della carta per accederne ai dettagli.

```
function TableApp(props){
    const{mukData} = props;

    const mukTr = mukData.map((muk) =>{
        return(
            <tr key={muk.numb}>
                <td>
                    <NavLink to={`/Appearances/${muk.numb}`}>
                        <img className={styleT.imgTable} onError={(event) => mukDefaultImg(event)}
                            src={muk.image} alt={muk.name}/>
                    </NavLink>
                </td>
                <td>
                    {muk.series}
                </td>
                <td>
                    {muk.set}
                </td>
                <td>
                    {muk.releaseDate}
                </td>
            </tr>
        );
    });

    return(
        <Table className={styleT.appTable}>
            <thead>
                <tr>
                    <th>Image</th>
                    <th>Series</th>
                    <th>Set</th>
                    <th>Release</th>
                </tr>
            </thead>
            <tbody>
            {mukTr}
            </tbody>
        </Table>
    );
```

# 3 View

## 3.1 App

App è il componente, stateless, generico dell'applicazione, utilizzato per inizializzarla. Al su interno avviene il processo di routing, che permette la navigazione tra le pagine, grazie ai componenti *{BrowserRouter, Routes, Route}* di React-router-dom.

Da questo componente, inoltre, vengono definite un oggetto "nav" e dei link usati nel footer; questi verranno passati come props al componente "Template".

```jsx
import React from "react";
import Template from "../../Components/Template/Template";
import Logo from "../../Assets/Images/Muk_Icon.png"
import Home from "../Home/Home";
import Appearances from "../Muk/Appearances";
import Info from "../Info/Info";
import Error from "../404/404";
import {BrowserRouter, Routes, Route} from "react-router-dom";
import MukDetails from "../Detail/MukDetails";
import  "../../index.css"

function App() {
    const nav = [
        {url: "/", text: "Home", exact: true},
        {url: "/Appearances", text: "Appearances", exact: true},
        {url: "/info", text: "Info", exact: true}
    ];
  return (
    <BrowserRouter>
      <Template
          footerCourseName = "Tecnologie e Applicazioni dei Sistemi Distribuiti"
          footerCourseLink = "https://elearning.unimib.it/course/info.php?id=44672"
          navItems={nav}
          logo={Logo}>
          <Routes>
              <Route path="/" element={<Home />}/>
              <Route path="/Appearances" element={<Appearances/>}/>
              <Route path="/Appearances/:number" element={<MukDetails/>}/>
              <Route path="/info" element={<Info />}/>
              <Route path="*" element={<Error/>}/>
          </Routes>
      </Template>
    </BrowserRouter>
  );
}

export default App;
```

## 3.2 Home

Il componente stateless “Home” visualizza la “landing page” del sito.

È probabilmente il componente più semplice di tutta l’applicazione e serve solamente a dare una breve introduzione ai contenuti del sito.

## 3.3 Appearances

“Appearances” è un componente stateful che mostra le varie apparizioni di Muk nel gioco di carte. Come detto nell’introduzione, è possibile visualizzare le carte in un formato che potremmo chiamare “didascalico”, o in formato tabellare, grazie ai componenti “TableApp” e “CardGrid”; si può passare da una visualizzazione all’altra grazie all’uso congiunto di una funzione “useState”, un rendering condizionale e un bottone di switch che gestisce lo stato.

```
import React, {useEffect, useState} from 'react';
import 'bootstrap/dist/css/bootstrap.min.css';
import clsx from "clsx";
import Container from "react-bootstrap/Container";
import Row from "react-bootstrap/Row";
import Col from "react-bootstrap/Col";
import mukData from "../../Assets/Data/Muk.json"
import CardGrid from "../../Components/CardGrid/CardGrid";
import TableApp from "../../Components/TableApp/TableApp";
import styleAp from "./Appearances.module.css"

function Appearances(){
    const [displayGrid, setDisplayGrid] = useState("true")
```

All'inizio l'utente vedrà la visualizzazione a carte poiché definita dallo stato true, che è quello standard di "displayGrid".

```
<Row>
   {displayGrid ? <CardGrid mukData={filteredList} col={{xs:1, sm:2, md:3, lg:3, xl:4}} /> : <TableApp mukData={filteredList}/>}
</Row>
```

Nel componente è stato anche inserito un sistema di filtraggio per serie.

Vengono definiti due stati: uno controlla che serie è stata scelta nel dropdown (selectedSeries), mentre l'altro gestisce le modifiche alla lista di carte e viene passato come prop.

La funzione filterBySeries consiste nel filtro vero e proprio, filtrando la lista dei personaggi in base alla voce del menù dropdown selezionata dall'utente con le seguenti scelte:

- L'if iniziale evita di filtrare la lista dei personaggi (filteredData) quando selectedSeries corrisponde ad una stringa vuota. In questo caso, infatti, non è necessario eseguire una funzione filter e viene quindi restituita la lista completa dei personaggi (filteredData).
- Se selectedSeeries non è una vuota, ovvero se l'utente seleziona una voce del dropdown, viene eseguita la funzione filter che restituisce solo gli oggetti dell'array filteredData che hanno come valore della proprietà "series" lo stesso valore di selectedSeries. La funzione restituiscequindi il nuovo array con i personaggi filtrati.

Viene successivamente utilizzato un hook "useEEffect", che chiama la funzione sopra descritta, per aggiornare la lista di carte. . Inoltre, come dipendenza di "useEffect" viene passato lo stato selectedSeries, in modo tale che la funzione di callback venga

eseguita ogni volta che o ogni volta che l'utente seleziona una nuova voce del menù dropdown.

Infine è presente una funzione "onChange", che cambia la serie alla selezione dell'utente.

```
const [filteredList, setFilteredList] = useState(mukData);
const [selectedSeries, setSelectedSeries] = useState("");
function handleSeriesChange(event) {
    setSelectedSeries(event.target.value);
}

function filterBySeries(filteredData) {
    if (!selectedSeries) {
        return filteredData;
    }
    const filteredMukCards = filteredData.filter(
        (mukCard) => mukCard.series === selectedSeries
    );
    return filteredMukCards;
}

useEffect(() => {
        let filteredData = filterBySeries(mukData);
        setFilteredList(filteredData);
    },
    [selectedSeries]);
```

```
<Col md={2}>
    <div className="d-flex justify-content-between align-items-center mb-5">
        <select
            id="series-input"
            className={styleAp.seriesSelect}
            value={selectedSeries}
            onChange={handleSeriesChange}
        >
            <option value="">Filter by series</option>
            <option value="Base">Base</option>
            <option value="EX">EX</option>
            <option value="Diamond & Pearl">Diamond & Pearl</option>
            <option value="HeartGold & SoulSilver">HeartGold & SoulSilver</option>
            <option value="Platinum">Platinum</option>
            <option value="Black & White">Black & White</option>
            <option value="Sun & Moon">Sun & Moon</option>
            <option value="Sword & Shield">Sword & Shield</option>
            <option value="Scarlet & Violet">Scarlet & Violet</option>
        </select>
    </div>
```

## 3.4 Details

“MukDetails” è un componente stateful che descrive alcuni dei dettagli delle carte.

Prima di chiamare l’API per ottenere i dati con cui popolare la pagina è stato necessario definire un modo per indicare la carta correntemente selezionata; a questo proposito è stato utilizzato un key proveniente dal file JSON già usato in precedenza.

per ottenere il personaggio corrente viene eseguita la funzione filter sul file JSON in modo da restituire la carta il cui id ha lo stesso valore dell’id della pagina corrente; questo id viene successivamente trasformato da stringa a intero attraverso l’uso della funzione “parseInt”. Una volta definito il giusto id è stata chiamata l’API attraverso l’hook “useEffect” e la funzione “fetch”, abbinando la richiesta all’id corrente. È stata, inoltre, una dipendenza in base all’id, che fa in modo che al cambiamento di quest’ultima venga rieseguita la fetch.

```jsx
import { useParams } from 'react-router-dom';
import { NavLink } from 'react-router-dom';
import mukData from "../../Assets/Data/Muk.json"
import mukDefaultImg from "../../Utility/Utility";
import styleD from "./Mukdetails.module.css"

function MukDetails () {
    const {number: noteNumber} = useParams();
    const mukId = parseInt(noteNumber);
    const currentMuk = mukData.filter((muk) => muk.numb === mukId)[0];
    const [mukCardInfo, setMukCardInfo]= useState([]);

    useEffect(() => {
        let isMounted = true;
        fetch(`https://api.tcgdex.net/v2/en/cards/${currentMuk.id}`)
            .then(res => res.json())
            .then(res => {
                if (isMounted)
                    setMukCardInfo(res);
            })
            .catch((error) => console.log("Error"+error));

        return () => {
            isMounted = false;
        }
    }, [currentMuk.id]);
```

La definizone dell'id corrente ha permesso anche la creazione di un sistema di navigazione tra le pagine attraverso dei bottoni.

```jsx
return(
<>
    <Container className="d-flex justify-content-start mt-3 mb-4 mb-md-0">
        <NavLink to={`/Appearances`}>
            <button type="button" className={styleD.bottone}>⮌  Back</button>
        </NavLink>
    </Container>
        <h1 className="text-center my-4">Card details</h1>
    <Container >
        <div className="d-flex justify-content-evenly mb-4">
            {mukId - 1 !== 0 &&
                <NavLink className={styleD.bottone} to={`/Appearances/${mukId - 1}`}>⮜ Previous</NavLink>
            }
            {mukId + 1 <= Object.keys(mukData).length &&
                <NavLink className={styleD.bottone} to={`/Appearances/${mukId + 1}`}>Next ⮞</NavLink>
            }
        </div>
```

## 3.5 Info

Info è un componente stateful che visualizza le informazioni sul Muk come Pokemon.

Questo componente è concepito in tre parti:

- Una "anagrafica" di Muk;

- Immagini di Muk;
- Informazioni su Muk.

La parte anagrafica è stata pensata per raccogliere dati sul personaggio provenienti da “PokeAPI”, in questo caso altezza, peso, abilità e statistiche. Dopo aver disposto due funzioni di stato per la gestione delle due chiamate all’API(sono due visto che Muk presenta due forme con caratteristiche diverse), svolte con una funzione “fetch”. A differenza della sezione “Details” , qui la chiamata è statica e non è dipendente. I dati ricevuti dall’API verranno poi smistati nelle costanti che definiscono le due forme.

```
function Info(){
    const [mukInfo, setMukInfo]= useState([]);
    const [aMukInfo, setAMukInfo]= useState([]);
    const [mukVersion, setMukVersion] = useState(false);
    const [toggleShiny, setToggleShiny] = useState(true);

    useEffect(() => {
        let isMounted = true;
        fetch(`https://pokeapi.co/api/v2/pokemon/muk`)
            .then(res => res.json())
            .then(res => {
                if (isMounted)
                    setMukInfo(res);
            })
            .catch((error) => console.log("Error"+error));

        return () => {
            isMounted = false;
        }
    }, []);

    useEffect(() => {
        let isMounted = true;
        fetch(`https://pokeapi.co/api/v2/pokemon/muk-alola`)
            .then(res => res.json())
            .then(res => {
                if (isMounted)
                    setAMukInfo(res);
            })
            .catch((error) => console.log("Error"+error));

        return () => {
            isMounted = false;
        }
    }, []);
```

La sezione immagini è stata pensata per mostrare tutte le forme di Muk, per questo motivo è stato utilizzato più volte un rendering condizionale. Il primo permette, con un bottone, l’alternarsi tra la forma base e la forma di Alola.

```
const Kanto = (
    <Row>
        <Col md={6}>
            {toggleShiny? <img onClick={() => setToggleShiny(false)} className={stylei.img1} src={Muk} alt="Muk"/> : <img onClick={() => setToggleShiny(true)} className={stylei.img1} src={
                <button
                    type="button"
                    className={stylei.bottone}
                    onClick={() => setMukVersion(false)}>
                    Switch to Alolan version
                </button>
        </Col>
        <Col md={6}>
            <Row className={stylei.mukInfoFrame}>
                {mukInfo.height &&
                    <Col >
                        <p><strong>Height:</strong> {mukInfo.height} m</p> {/*rendering condizionale alolan}*/}
                    </Col>
                }
                {mukInfo.weight &&
                    <Col>
                        <p><strong>Weight:</strong> {mukInfo.weight} kg</p>
                    </Col>
                }
            </Row>
            <Row className={stylei.mukInfoFrame}>
                {mukInfo.abilities &&
                    <Col>
                        <strong className={stylei.ulTitle}>Abilities</strong>
                        <ul>
                            {mukInfo.abilities.map((abilityInfo) => {
                                    return <li key={abilityInfo.ability.name}>{removeDashesAndUnderscores(abilityInfo.ability.name)}</li>
                                }
                            )}
                        </ul>
                    </Col>
                }
                {mukInfo.stats &&
```

Il rendering condizionale in questo caso è presente dopo il return della funzione.

```
return(
    <>
        <Container>
            {mukVersion ? Kanto : Alola}
```

Un secondo rendering condizionale è stato usato all'interno delle costanti per le forme e viene utilizzata per passare alla versione shiny del Pokémon. Il toggle avviene cliccando sull'immagine.

```
{toggleShiny?<img onClick={() => setToggleShiny(false)} className={stylei.img1} src={alolanMuk} alt="Alolan-Muk"/> : <img onClick={() => setToggleShiny(true)} className={stylei.img1} src={shinyAlola}
```

L'ultima parte della pagina è didascalica, con informazioni generali su Muk.

## 4 Sviluppi futuri

Il progetto per quanto, a grosso modo, completo ha comunque ampi margini di miglioramento.

Tra i perfezionamenti pensati per eventuali successive release :

- Miglioramento generale dell'estetica del sito;
- Aggiunta di un'opzione di sorting per data  di rilascio nella sezione "Appearances"; questa funzionalità era già stata pensata per questa release, ma per motivi di tempo è stata scartata;

- Aggiunta di informazioni, raffinare quelle attuali e risolvere un problema con le API nella sezione “Info”;
- Aggiunta di più apparizioni, non solo nel TCG.