Manuale

DnD Wiki

Versione	Data	Redazione	Verifica	Messaggio
00	23.09.2025	A.Bettini	A.Bettini	Emissione iniziale

Sommario

1 DnD WIKI

This is a **React Application** that works as a Dungeon and Dragons Wiki to use during game sessions.

1.1 How does this work

The Application allow to select a category and search an item in that category thanks to input fields.

All the information are taken from the DnD API at this link: "https://www.dnd5eapi.co/api/".

1.2 Input Management

The input is taken from an input textfield and a select field:

```
<div className='top-banner'>
    <h1>API Test</h1>
    <form onSubmit={e => {
        e.preventDefault(); // Prevents page reload
        fetchContent()
    }}>
        <select className='dnd-input' name="category" id="category" onChange={handleCategoryChange}>
            <option value="spells">Spells</option>
            ...
            <option value="rule-sections">Rule Sections</option>
        </select>
        <input className='dnd-input' type="text" autoFocus name="search" id="search" required onChar
        <br></br>
        <br></br>
        <button className='dnd-button'>Search</button>
    </form>
</div>
```

1.3 Fetching Data

When the form is confirmed with either **[Enter]** or by pressing the button **[Search]** the function **fetchContent()** is called:

```
function fetchContent(){
    fetch(`https://www.dnd5eapi.co/api/${category}/${item}`)
    .then((res) => res.json())
    .then((data) => {
        setResult(data)
        setSuccess(true)
        console.log(data);
    })
    .catch((error) => {
        setSuccess(false)
        console.error('Error fetching data:', error);
    });
};
```

This function calls the **javascript function fetch()** to call the api and retrieve the data. It then checks if the call was succesfull with **.then** or if an error was catched with **.catch**

If the api call was succesfull the data read are set in result useState which is declared at the top of the script:

```
const [result, setResult] = useState()
```

The success useState is set true when the data are read succesfully, false if there was an error.

1.4 Showing Data

Based on the value of the success useState the **let resultContent** content is defined:

```
let resultContent;

  if (success) {
  resultContent = (
          <div className='container'>
              <div className="result-showcase">
                  <h2>{result.name}</h2>
                  <JSONViewer data={result} link={`/api/2014/${category}/${item}`}/>
                  <img src={prefix + result.image} alt={result.name} />
              </div>

              {sideSuccess && (
                  <div className="side-div">
                      <h2>{sideResult.name}</h2>
                      <button className='top-right-button' onClick={() => {
                          closeSideContent()
                      }}>X</button>
                      <JSONViewer data={sideResult} link={sideLabel}/>
                      <img src={prefix + sideResult.image} alt={sideResult.name} />
                  </div>
              )}
          </div>
      );
  } else {
      resultContent = (
          <>
              <p className='custom-error'> Input was either non-existent or null </p>
              <br></br>
              <p className='custom-error'> Insert a correct input to search succesfully </p>
          </>
      )
  };
```

This **resultContent** is then showed in the return:

```
{resultContent}
```

1.5 JSONViewer

The **JSONViewer** is a Component declared inside the MainMenu Component and it is used to show JSON in a pretty and readable way.

```
type JSONViewerProps = {
readonly data: any;
readonly link: string;
};

function JSONViewer({ data, link }: JSONViewerProps) {
    if (typeof data === 'object' && data !== null) {
        return (
```

```
            <ul>
                {Object.entries(data).map(([key, value]) => {
                    if(key === 'url' && value !== link){
                        return (
                            <li key={key}>
                                <strong>{key}:</strong>
                                <button className='link-button' onClick={() => {
                                    fetchSideContent(String(value))
                                }}>
                                    {String(value)}
                                </button>
                            </li>
                        )
                    } else {
                        return (
                            <li key={key}>
                                <strong>{key}:</strong> <JSONViewer data={value} link =''/>
                            </li>
                        )
                    };
                })}
            </ul>
            )
        } else {
            return <span>{String(data)}</span>;
        }
    }
```

1.6 Side Content

Inside the main content will occasionally appear some **clickable links** routing to other content of the API. The click action of these links is dealed in the JSONViewer which finds them and sets them as clickable buttons. When clicked, the function **fetchSideContent** is called, passing in input a string value representing the given path.

```
    function fetchSideContent(label: string){
        fetch(`https://www.dnd5eapi.co${label}`)
          .then((res) => res.json())
          .then((data) => {
            setSideResult(data)
            setSideSuccess(true)
            setSideLabel(label)
            console.log(data);
          })
          .catch((error) => {
            setSideSuccess(false)
            console.error('Error fetching data:', error);
          });
    };
```

This function executes another call to the API equal to the one in fetchContent(). When succesfull the function sets the **sideResult** data, the **sideSuccess** and the **sideLabel** value which are all useStates defined in the MainMenu Component. So now, thanks to the if in the letContent return content, a **side content** containing the data retrieved from the call is shown next to the main content.

In this sideContent there is also a close button positioned on the top right which closes the sideContent calling

the function **closeSideContent()**:

```
function closeSideContent(){
    //closes side content
    setSideSuccess(false)
}
```

2 DnD Character Builder

To the DnD Wiki Application I now added a new section with a simple character builder where the user can create their character and then see it and consult the values needed.

2.1 Character Interface

The character parameteres are defined in this **interface**:

```
interface Character {
    name: string;
    playerName: string;
    level: string;
    class: string;
    race: string;
    alignment: string;
    statistics: {
        hp: string;
        ac: string;
        profBonus: string;
    };
    ability: {
        str: string;
        dex: string;
        con: string;
        int: string;
        wis: string;
        cha: string;
    };
}
```

2.2 Create the Character

When the user first opens up the page he will see a message saying there is no character saved and a button to start creating one This is managed in a similar way to the showed content in MainMenu, with a **let characterContent**, we check if the created useState is true (which is saved in localStorage). If it is true the character's information is showed. If it is false the button is showed. This happens thanks to this code segment:

```
let characterContent;

if(created){
    characterContent = (
        <div className='top-banner'>
            <h1>Your Character</h1>
            <p><strong>Player:</strong> {character.playerName}</p>
            <p><strong>Name:</strong> {character.name}</p>
            <p><strong>Class:</strong> {character.class}</p>
            <p><strong>Race:</strong>  {character.race}</p>
            <p><strong>Level:</strong>  {character.level}</p>
            <p><strong>Alignment:</strong> {character.alignment}</p>
            <p><strong>STATISTICS:</strong></p>
            <p><strong>HP:</strong> {character.statistics.hp}</p>
            <p><strong>AC:</strong> {character.statistics.ac}</p>
            <p><strong>Proficiency Bonus:</strong> {character.statistics.profBonus}</p>
            <p><strong>ABILITIES SCORES:</strong></p>
```

```
                <strong> STR:</strong> {character.ability.str}   
                <strong> DEX:</strong> {character.ability.dex}   
                <strong> CON:</strong> {character.ability.con}   
                <strong> INT:</strong> {character.ability.int}   
                <strong> WIS:</strong> {character.ability.wis}   
                <strong> CHA:</strong> {character.ability.cha}   
                <br></br>
                <br></br>
                <button className='dnd-button' onClick={() => {
                    StartCreation();
                }}>Create New Character</button>
                <br></br>
                <br></br>
            </div>
        )
    }
    else if(!isCreating){
        characterContent = (
            <>
                <br></br>
                <p className='custom-error'>No character created</p>
                <button className='dnd-button' onClick={() => {
                    StartCreation()
                }}>Create a new Character</button>
            </>
        )
    }
```

When the user clicks the create button the function **StartCreation** is called:

```
const [isCreating, setIsCreating] = useState(false);

function StartCreation(){
    setIsCreating(true);
    setCreated(false);
}
```

This function set **isCreating** useState to true e created to false. Then with this code segment the same thing happens and isCreating being true the form to create the character is showed:

```
if(isCreating){
    characterCreation = (
        <div className='top-banner'>
            <h1>Build your own Character</h1>
            <form onSubmit={e => {
                e.preventDefault(); // Prevents page reload
                createCharacter()
            }}>
                ... input fields ...
            </form>
        </div>
    )
} else{
    characterCreation = (
        <>
        </>
```

```
        )
    }
```

When the form is submitted the function **createCharacter()** is called:

```
function createCharacter(){
    setCreated(true);
    setIsCreating(false);
}
```

This set created to true, allowing the segment showed before to render the **character's information and data**.

2.3 Saving and Loading

The character is obviously both **saved and loaded** thanks to useStates and useEffects with localStorage:

```
const [character, setCharacter] = useState<Character>(() => {
    const saved = localStorage.getItem("character");
    return saved ? JSON.parse(saved) : {
        name: '',
        playerName: '',
        level: '',
        class: '',
        race: '',
        alignment: '',
        statistics: { hp: '0', ac: '0', profBonus: '0' },
        ability: { str: '0', dex: '0', con: '0', int: '0', wis: '0', cha: '0' },
    };
});

useEffect(() => {
    localStorage.setItem("character", JSON.stringify(character));
}, [character]);
```

3 DnD Take Notes

A new page in this DnD React Application is the Take Notes page, loaded from the NoteTaker Component.

3.1 Take Notes

This page allows the user to **take notes** during the sessions and see them written besides instantly. In the page there are 2 divs, on the left and on the right. The left one shows the notes from the useState while the right one has a **textarea** for the input.

```
<h1>Take notes of your session!</h1>
  <div className='notes-container'>
    <div className='left-notes-div'>
      <div>{notes}</div>
    </div>
    <div className='right-notes-div'>
      <textarea
        rows={20}
        className='dnd-input-textarea'
        placeholder='notes...'
        autoFocus
        value={notes}
        onChange={handleChange}
      />
    </div>
  </div>
```

The notes are saved and loaded thank to **localStorage** as everything else:

```
const [notes, setNotes] = useState(() => {
    const saved = localStorage.getItem("notes");
    return saved || '';
});

useEffect(() => {
    localStorage.setItem("notes", notes);
}, [notes]);
```