User-Blog-Gallery

https://github.com/alienbrains/User-Blog-Gallery/

Src

App.js

```
import React from "react";
import UserGallery from "./Components/UserGallery/index";

import "./App.css";

function App() {
 return <UserGallery />;
}

export default App;
```

This is the initial App.js Component which is a functional component.

Components

Components/UserGallery/index.js

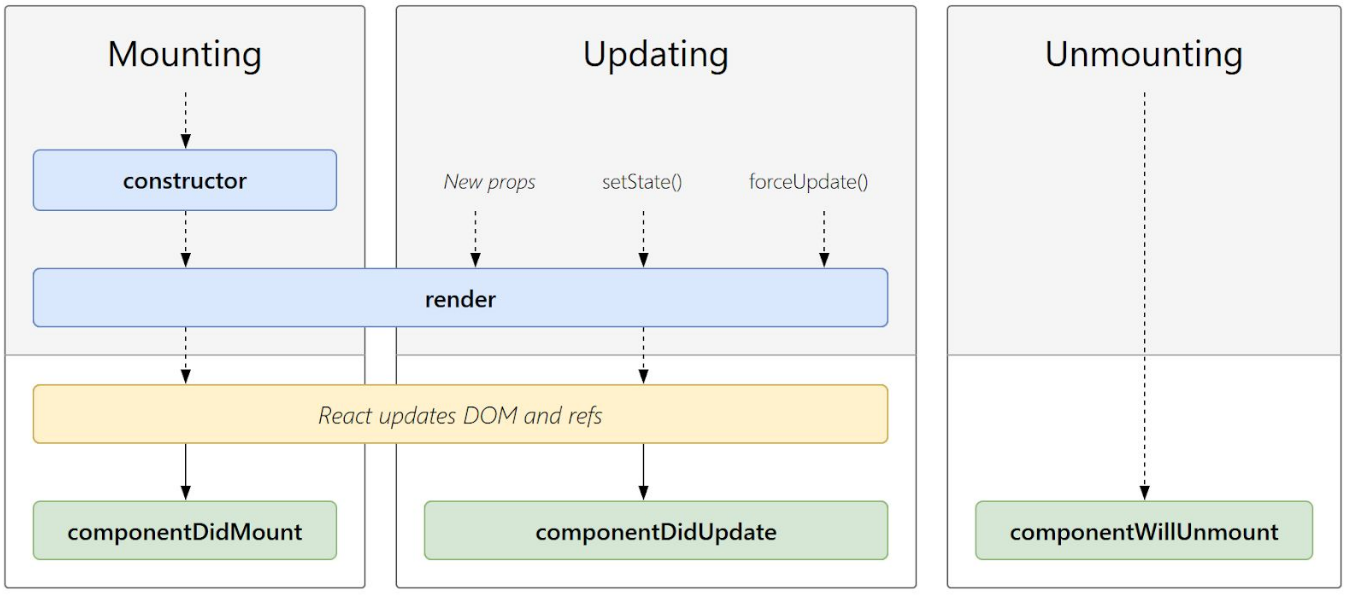
```
import React from "react";
import UserCard from "../UserCard/index";

import "./style.css";

class UserGallery extends React.Component {
 constructor(props) {
   super(props);
   this.state = {
     users: [],
     isLoading: false,
     currentPage: 0
   };
 }

 //  lifecycle method of React which is called for only once after the
first render method is called
 componentDidMount = () => {
   //  fetching users when the page loads
   this.fetchUsers();
 };

 fetchUsers = () => {
   //  setting which page to fetch
   const pageToFetch = this.state.currentPage + 1;

   //  url from which data should be fetched
   const url = `https://reqres.in/api/users?page=${pageToFetch}`;

   //  updating the loading status to true to show loading
   this.setLoading(true);
```

```
  //  starting to fetch data
  fetch(url, {
    method: "GET"
  })
    .then((response) => {
      return response.json();
    })
    .then((result) => {
      //  copying the old users along with the new users
      const prevUsers = [...this.state.users, ...result.data];

      //  updating the user list and currentPage
      this.setState({
        users: prevUsers,
        currentPage: pageToFetch
      });

      //  updating the loading status to false to hide loading
      this.setLoading(false);
    })
    .catch((error) => {
      //  updating the loading status to false to hide loading
      this.setLoading(false);
    });
};

setLoading = (status) => {
  this.setState({
    isLoading: status
  });
};

render = () => {
  return (
    <div className="container">
      <p className="title">------ User Gallery--------</p>
      <div className="show-area">
        {this.state.users.map((user) => {
          return (
            <UserCard
              key={user.id}
              picUrl={user.avatar}
              firstName={user.first_name}
              lastName={user.last_name}
              email={user.email}
            />
          );
        })}
      </div>
      {this.state.isLoading ? (
        <span className="loading-text">Loading ...</span>
      ) : (
        <button className="load-btn" onClick={this.fetchUsers}>
          Load More
        </button>
      )}
    </div>
  );
 };
}

export default UserGallery;
```

The state consists of three data elements

1. users - An array to store the data of the users which we are adding. Initially an empty array.
2. isLoading - A boolean value, which is set to **true** when we start fetching the user data, and set to **false** when the data fetching is complete. Initially set to **false**.
3. currentPage - A number which is initially 0 so as to fetch data from the first page of the api. When it is increased it fetches data from the next next page of the api call.

componentDidMount()
This is a react lifecycle method, this method is called only once after the first **render()** is executed.



As you can see from here,
The first thing that happens in **React** is Mounting.

The first function which gets called is the **constructor** which assigns initial values to the state variables.

The second function to get called is the **render(),** which displays the initial view which we have written inside the **return** of the **render** function.

The third function which gets executed is the **componentDidMount** , in simple english it means that this function is executed after the first mounting takes place, that's why we say component**Did**Mount.

After the **componentDidMount(),** again the **render()** function is executed.

In this project the **componentDidMount()** calls the function **this.fetchUsers().**

this.fetchUsers()
This function at the first line stores the page value that is to be fetched in the variable **pageToFetch**.
In the next line it sets the **url** from where the data is to be fetched along with the page number. To set the page number we use string interpolation, **`page=${pageToFetch}`** , so as to dynamically set the page number every time we change it.
Now, since we are about to start to fetch data from the server, we set the state of **isLoading** to true by calling the function **this.setLoading** and passing **true** as an argument.

Now, we call the method **fetch()** in JavaScript. This function takes two arguments. One which consists of the **url** from where the data will be fetched. The other consists of the type defined by the key **method.** The fetch responds with a **Promise,** we will talk about it later**.**

This function is asynchronous in nature, and is used to make **XMLHttpRequest.**

XMLHttpRequest is used to send data from the server side to the client side or vice versa.
Here we use the **method: "GET"** in the fetch function, this indicates that the client wants data from the server side.
There is another **method: "POST"** which indicates that the client wants to submit data to the server side.

The server takes some time to return the data that we ask for, therefore we have to wait over there. When the waiting is over and the response is a success we execute the **.then()** function.

Thus we **fetch(something).then(we wait for the response to be a success)** this is the syntax.

Inside the **then** function we receive the **response** which we write like a fat arrow function.

fetch (something).then ((response) => {
return response.json()
})

We return the response in the json() format so that the data is readable in the client side.

This response is **then** stored as a result so that we can make the necessary changes in our data.

fetch(something)
.then(we wait for the response which we return in json format)
.then(we receive the desired result which we want to use)

```
.then((result) => {
       //  copying the old users along with the new users
       const prevUsers = [...this.state.users, ...result.data];

       //  updating the user list and currentPage
       this.setState({
         users: prevUsers,
         currentPage: pageToFetch
       });

       //  updating the loading status to false to hide loading
       this.setLoading(false);
     })
```

Now, in this **.then()** function we receive the result .

This result is an object where the users are present in the key **result.data .** We copy the previous value of **this.state.users** and the current **result.data,** using the spread operator (or the “three dots”), in a new array named **prevUsers**.
We then update the value of the state variables, **users** with **prevUsers** and **currentPage** with **pageToFetch,** using the **setState** function.
After that we set the state of **isLoading** to false, by calling the function **this.setLoading** and passing the argument false.

After this entire process takes place, we will be able to see the users in our page as the component re-renders.

```
.catch((error) => {
       //  updating the loading status to false to hide loading
       this.setLoading(false);
     });
```

The **.catch()** function in this entire **fetch** process will only get executed, if for some reason we are unable to receive the data from the server side due to some error, maybe internet connectivity or some mistake.
Thus,

fetch(something)
.then(we wait for the response which we return in json format)
.then(we receive the desired result which we want to use)
.catch(if there are any error which causes a failure in this process)

When we click the **Load More** button we call the function **this.fetchUsers,** but now we call the users with an updated page number.
We then again set the new page number using string interpolation.
We set the state of **isLoading** to true.
Again, fetch the data from the server side but with the updated page number, so now in this case we update the state with new users.

Components/UserCard/index.js

```
import React from "react";

import "./style.css";
const UserCard = (props) => {
 return (
   <div className="user-card">
     <img className="user-img" src={props.picUrl} />

     <span>
       {props.firstName} {props.lastName}
     </span>

     <span>{props.email}</span>
   </div>
 );
};

export default UserCard;
```

The **UserCard** component accepts **props** passed by the parent component **UserGallery** the <img /> tag displays the image using the **src** attribute, which is a string, stored in **prop.picUrl .**
In the <span /> tag we display the name using the props, **prop.firstName** for the first name and **props.lastName** for the last name.
We also display the email address using **props.email** in another <span/> tag.

Promises

The **Promise** object represents the eventual completion (or failure) of an asynchronous operation, and its resulting value. Here, fetch() returns a **promise** which is an object. Depending on whether the promise is **resolved** (which means the response is a success) or **rejected** (which means the response failed), the corresponding functions get executed. If it is a success we execute **.then()** if a failure we execute **.catch() .**

JSON format

JSON or JavaScript Object Notation, is a lightweight format of storing data and transporting it. It is used to transfer data from the server side to the client side. It is very easy to understand.
Example :

```
{
 "userName1": "Player1",
 "userName2": "Player2"
}
```