**Infinite scroll loading using**

**Intersection Observer and React Hooks**

In this application, I have created a function component that uses the Intersection Observer interface and React hooks and another function component that loads images from a specific API that retrieves data with a given size. The scope of the application is to create an infinite scrolling with react that lazily loads images.

**React**

React is a JavaScript library, which offers the user the possibility of creating user interfaces with declarative code. It aims to build complex and fast Web applications using JavaScript. It is a highly flexible and efficient JavaScript library. It displays information by using components. The concept of a component can be defined as a reusable UI.

Using components, React encourages the user to structure the application into separate blocks. Components act independently of each other, but in the case of complex UIs, they can interact and pass rich data through to other components with the use of props. Since there is a unidirectional data flow between components, the props are sent only from parent to child.

**Intersection Observer API**

The Intersection Observer API allows you to asynchronously monitor changes in a target element's intersection with an ancestor element or with the viewport of a top-level document.

Detecting an element's visibility, or the relative visibility of two items concerning each other has been a tough task for which solutions have been unreliable and prone to slowing down the browser and the sites the user is visiting. The demand for this type of information has increased as the internet has evolved. Information about intersections is required for a variety of reasons, including:

- Images Lazily-loading as a page is scrolled

- Implementing "infinite scrolling" websites, where more and more content is loaded and rendered as you scroll so that the user doesn't have to flip through pages which is the case of pagination

- Animation processes available to the user at some part of the page only.

**React Hooks**

React provides some built-in hooks that are functions which are dealing with the state and lifecycle features of function components. Hooks cannot be used inside of React classes.

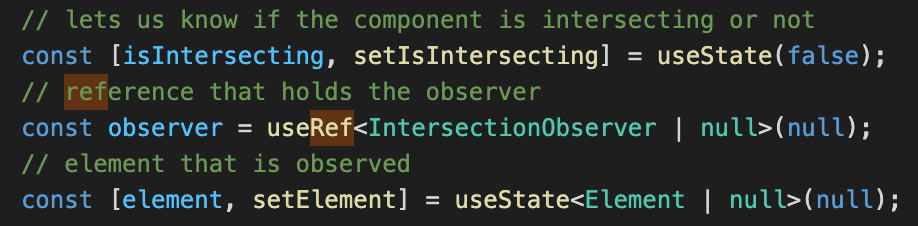
Besides those built-in hooks, we can create other hooks that we may need in our applications. Such examples will be found in the presented application.

Some of the built-in hooks that were used in this app are: useRef, useEffect, useState.

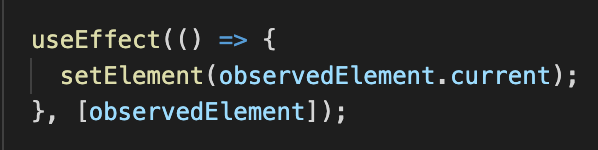
**useIntersectionObserver**

One of the most important parts of the provided application is the function component **useIntersectionObserver**. In this component, we are using Intersection Observer to monitor elements, and check if those elements are intersecting with the viewport. This hook will be later used to create infinite scrolling.

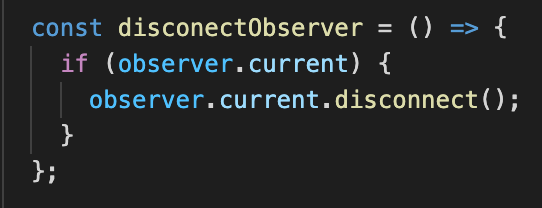
The hook that was created contains a state that will let us know if the component is intersecting or not. Another thing that is needed is the reference that holds our observer, the Intersection Observer. We also need an element that is considered as a target, and this one is a parameter and also a state in our function component.



Whenever the observed element changes, we need to update the target element we are looking for. For this reason, we use the useEffect hook provided by React.



 Then we need to create the observer that observers the target element. We use again the useEffect hook, and check first if the element is null. The function component that we have created takes as parameters that element to be observed, and a boolean parameter forward that suggests if we want the observer to be disconnected after the element was found in the viewport. And of course, after the element was found, and we don’t want to keep the observer anymore, we can disconnect the observer.



**Load Images**

The second important component that was created in this project is an Image loader. Since the purpose of the application is to create infinite scrolling, we want to load several images at a time, and when we reach the bottom of the page, we want to load another batch of images a.s.o.

Since we have built the useIntersectionObserver hook, we can use it now. In our case, we took batches of 5 images, but this variable can be set to any other value.

The LoadImages component is captured in the following image:

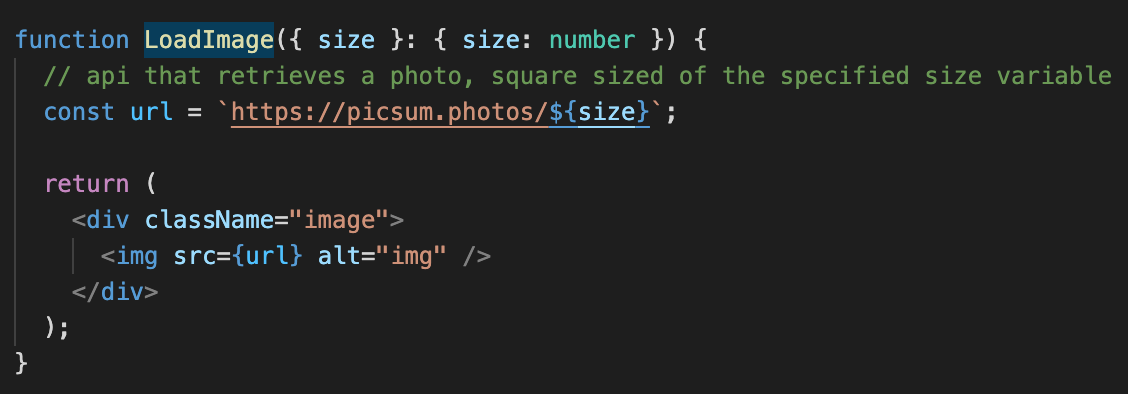


As we can see, there are a number of 5 images that are loaded at once. There is a ref on the loading div that should appear at the bottom of the page.

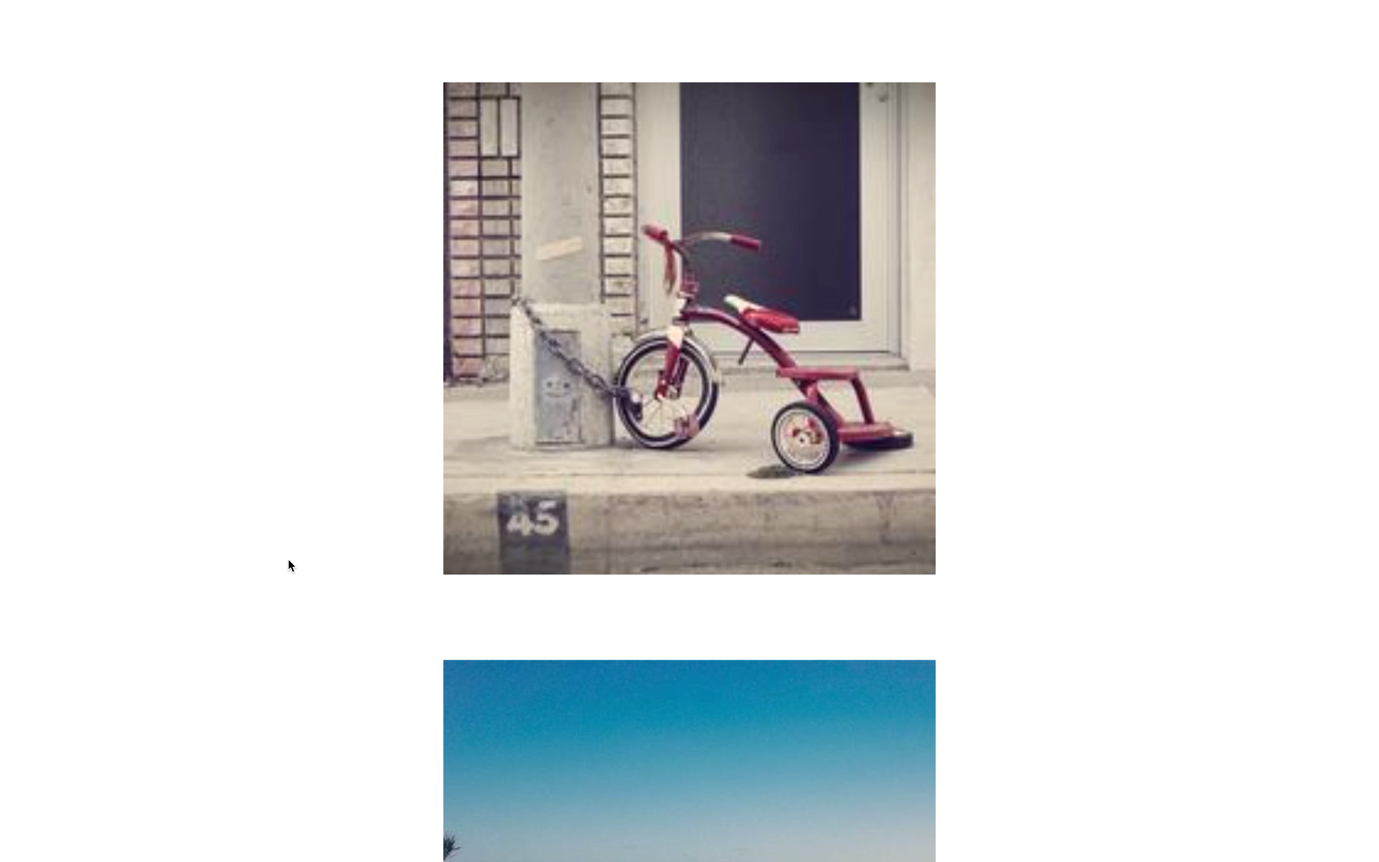
This ref is passed to the **useIntersectionObserver** hook, together with the true value which means that we want to keep the observer to observe further.

In order to construct the image array, I have created another component that makes a request to a public API that retrieves an image of a specified size and then adds the image in a div.

The provided API can retrieve squared photos if one length only is given, but it can also retrieve photos of other dimensions.



And this is how a react infinite scroll loading, using intersection observer and react hooks it is achieved.

 The final result of the presented application can be seen in the following video. Pay attention at the scroll bar ☺.