Project Report

Snapshot Gallery

Rehmat | 2110994813

24 October, 2023

Introduction

Project Overview

The goal of the React project is to employ React for fluid UI development in the creation of an interactive image gallery. One of its main goals is to give users an easy-to-use image management experience. Users can benefit from a dark mode option for improved visibility in addition to being able to dynamically add and remove photos. Utilizing technologies like React, JavaScript, JSX, CSS, and HTML, the project creates an application that is both aesthetically pleasing and easy to use.

Objectives

The main objectives of this project include:

1. **Implementing an Easy-to-Use Interface:** The project's main goal is to give users a simple way to interact with the picture gallery.
2. **Dynamic Image Management:** This feature enables flexible content management by enabling users to upload new images to the gallery and remove old ones.
3. **Dark Mode Integration:** For better visibility and aesthetics, the application has a dark mode option.

Technology used

The project employs a powerful tech stack to create a dynamic image gallery:

1. **React**: The project's main component, React is a JavaScript library well-known for creating dynamic user interfaces. It makes it possible to create reusable user interface components, which guarantees an effective and manageable codebase.

2. **JavaScript & JSX:** The foundation of the application's interaction is formed by JavaScript and JSX. Because of its flexibility, JavaScript may be used for sophisticated functionalities, and JSX makes it easier to integrate HTML-like syntax. When combined, they provide for smooth user interactions.

3. **Cascading Style Sheets, or CSS**, control the visual layout and guarantee a responsive and aesthetically beautiful design. In order to create an engaging user experience, CSS is essential for styling, positioning, and animating items.

4. **Hyper Text Markup Language (HTML)** Linguistics supplies the application's structural framework. It helps create a logical and user-friendly interface by arranging components and content. SEO and usability are improved by HTML's semantic structure.

The styling power of CSS, the reusability of React, the adaptability of JavaScript, and the structural integrity of HTML are all skillfully combined in this tech stack. They enable the project to produce an engaging picture gallery and a smooth user experience when combined.

Project Structure

The project exhibits a well-organized structure with distinct components, each contributing to specific functionalities:

App.js: Serving as the application's entry point, App.js manages the global state and orchestrates key components. It efficiently handles the list of images and controls the dark mode toggle, ensuring a seamless user experience.
DarkModeToggle.jsx: This reusable component facilitates the toggling between light and dark modes. It is strategically positioned for easy access, providing users with a visual indicator and functionality for customizing their viewing experience.
Gallery.jsx: Responsible for rendering the image gallery, Gallery.jsx offers users the ability to sort images by date or name. It integrates a user-friendly drag-and-drop feature for effortless addition of images, enhancing overall usability.
AddImage.jsx: Focused on image management, this component oversees the process of adding new images to the gallery. It incorporates a loading state to simulate an upload delay, ensuring a smooth and responsive user interaction.
Image.jsx: Representing individual images, Image.jsx showcases each image and offers a convenient removal option. This component enhances user control and personalization within the gallery.
ImageUploadValidation.jsx: Dedicated to validating uploaded images, this component ensures that only valid image files are accepted. It provides user-friendly error handling, enhancing the overall robustness and reliability of the application.

This structured approach to component organization fosters code modularity and readability, simplifying maintenance and future enhancements. Each component fulfills a distinct role, contributing to the overall functionality and user experience of the image gallery application.

Conclusion

In summary, the React-based dynamic picture gallery accomplishes its main goals with success. Users may easily manage their photographs thanks to a well thought-out component architecture and interface. The addition of dark mode improves accessibility and user personalization. Utilizing HTML, CSS, JSX, JavaScript, and React, the application blends style with functionality. Scalability and maintainability are guaranteed by the organized approach to component organization used in the project. All things considered, the picture gallery offers an engaging user experience by showcasing expertise in both technical implementation and user-centric design.

Code Screenshots and Analysis

App.js

my-app > src > JS App.js > ...

```
import React, { useState } from 'react';
import './App.css';
import Gallery from './Gallery';
import AddImage from './AddImage';
import DarkModeToggle from './DarkModeToggle';
import './DarkModeToggle.css';


Codeium: Refactor | Explain | Generate JSDoc
function App() {
  const [images, setImages] = useState([]);
  const [darkMode, setDarkMode] = useState(false);

  Codeium: Refactor | Explain | Generate JSDoc
  const handleAdd = (imageUrl) => {
    setImages([...images, { id: Date.now(), imageUrl,
    dateAdded: new Date(), name: `Image ${images.length
    + 1}` }]);
  };

  Codeium: Refactor | Explain | Generate JSDoc
  const handleRemove = (id) => {
    setImages(images.filter((img) => img.id !== id));
  };

  Codeium: Refactor | Explain | Generate JSDoc
  const toggleDarkMode = (mode) => {
    setDarkMode(mode);
  };

  return (
    <div className={`app ${darkMode ? 'dark-mode' : ''}`}
    >
      <DarkModeToggle toggleDarkMode={toggleDarkMode} />
      <h1>Snapshot Gallery</h1>
      <AddImage handleAdd={handleAdd} />
      <Gallery images={images} handleRemove=
      {handleRemove} handleAdd={handleAdd} />
    </div>
  );
}

export default App;
```

Description: The App.js file serves as the entry point for the application. It orchestrates the main components and manages the state of the application, including the list of images and the dark mode toggle.
Analysis: This component demonstrates effective state management and the integration of various components.

DarkModeToggle.jsx

my-app > src > JS DarkModeToggle.jsx > ...

```jsx
import React, { useState } from 'react';
import './DarkModeToggle.css'; // Import the updated CSS file

Codeium: Refactor | Explain | Generate JSDoc
const DarkModeToggle = ({ toggleDarkMode }) => {
  const [darkMode, setDarkMode] = useState(false);

  Codeium: Refactor | Explain | Generate JSDoc
  const handleClick = () => {
    setDarkMode(!darkMode);
    toggleDarkMode(!darkMode);
  };

  return (
    <div className={`dark-mode-toggle ${darkMode ? 'dark-mode' : ''}`} onClick={handleClick}>
      Change Mode
    </div>
  );
};

export default DarkModeToggle;
```

Description: This component handles the toggling of dark mode in the application. It is a reusable element that provides a visual indicator and

functionality for changing the application's theme.

Analysis: The DarkModeToggle component showcases the implementation of a user-friendly feature for changing the application's theme.

Gallery.jsx

my-app > src > JS Gallery.jsx > Gallery

```jsx
import React, { useEffect, useCallback, useState } from "react";
import Image from "./Image";
import "./Gallery.css";

Codeium: Refactor | Explain | Generate JSDoc
const Gallery = ({ images, handleRemove, handleAdd }) => {
  const [sortCriteria, setSortCriteria] = useState("default");
  const [sortedImages, setSortedImages] = useState([...images]);

  Codeium: Refactor | Explain | Generate JSDoc
  const handleSort = (criteria) => {
    setSortCriteria(criteria);
  };

  Codeium: Refactor | Explain | Generate JSDoc
  const handleDragOver = (e) => {
    e.preventDefault();
  };

  const handleDrop = useCallback(
    (e) => {
      e.preventDefault();

      const droppedFile = e.dataTransfer.files[0];
      const reader = new FileReader();

      Codeium: Refactor | Explain | Generate JSDoc
      reader.onload = (event) => {
        handleAdd(event.target.result);
      };

      reader.readAsDataURL(droppedFile);
    },
    [handleAdd]
```

```
);

useEffect(() => {
  Codeium: Refactor | Explain | Generate JSDoc
  const adjustImageDimensions = () => {
    const images = document.querySelectorAll(".image
    img");
    let maxHeight = 0;

    images.forEach((image) => {
      const aspectRatio = image.naturalWidth / image.
      naturalHeight;
      const newHeight = aspectRatio * 150;

      if (newHeight > maxHeight) {
        maxHeight = newHeight;
      }
    });

    images.forEach((image) => {
      const aspectRatio = image.naturalWidth / image.
      naturalHeight;
      if (aspectRatio === 1) {
        image.style.height = "auto";
        image.style.width = "auto";
      } else {
        image.style.height = `${maxHeight}px`;
        image.style.width = `${maxHeight / aspectRatio}
        px`;
      }
    });
  };

  adjustImageDimensions();
  window.addEventListener("resize",
  adjustImageDimensions);

```

```
  const gallery = document.querySelector(".gallery");

  gallery.addEventListener("dragover", handleDragOver);
  gallery.addEventListener("drop", handleDrop);

  return () => {
    window.removeEventListener("resize",
    adjustImageDimensions);

    gallery.removeEventListener("dragover",
    handleDragOver);
    gallery.removeEventListener("drop", handleDrop);
  };
}, [handleDrop]);

useEffect(() => {
  const sortedImages = [...images].map((image) =>
  ({ ...image }));

  switch (sortCriteria) {
    case "default":
      // No sorting required
      break;
    case "date":
      sortedImages.sort((a, b) => {
        return new Date(b.dateAdded) - new Date(a.
        dateAdded);
      });
      break;
    case "name":
      sortedImages.sort((a, b) => {
        return (a.name || "").localeCompare(b.name ||
        "");
      });
      break;
    default:
```

```
91            break;
92          default:
93            break;
94        }
95
96        setSortedImages([...sortedImages]);
97      }, [images, sortCriteria]);
98
99      return (
100       <div className="gallery">
101         <div className="sort-buttons">
102           <button onClick={() => handleSort("default")}
              >Default</button>
103           <button onClick={() => handleSort("date")}>By
              Date</button>
104           <button onClick={() => handleSort("name")}>By
              Name</button>
105         </div>
106         {sortedImages.map((image, index) => (
107           <Image
108             key={image.id}
109             image={image.imageUrl}
110
111             handleRemove={() => handleRemove(image.id)}
112           />
113         ))}
114       </div>
115     );
116   };
117
118   export default Gallery;
119
```

Description: The Gallery.jsx component is responsible for rendering the image gallery. It allows users to sort images by date or name and provides drag-and-drop functionality for adding images.

Analysis: This component highlights the use of grid layout and the integration of user interaction features.

AddImage.jsx

my-app > src > JS AddImage.jsx > ...

```jsx
import React, { useState } from 'react';
import ImageUploadValidation from './
ImageUploadValidation';
import LoadingState from './LoadingState';

Codeium: Refactor | Explain | Generate JSDoc
const AddImage = ({ handleAdd }) => {
  const [loading, setLoading] = useState(false);

  Codeium: Refactor | Explain | Generate JSDoc
  const handleAddImage = (imageUrl) => {
    setLoading(true);

    // Simulate a delay for the upload process
    setTimeout(() => {
      handleAdd(imageUrl);
      setLoading(false);
    }, 2000); // 2 seconds delay
  };

  return (
    <div>
      {loading ? (
        <LoadingState />
      ) : (
        <ImageUploadValidation handleAdd=
        {handleAddImage} />
      )}
    </div>
  );
};

export default AddImage;
```

Description: This component manages the process of adding new images to the gallery. It includes a loading state to simulate an upload delay.
Analysis: The AddImage component demonstrates the handling of asynchronous operations and user feedback during image upload.

Image.jsx

```
my-app > src > JS Image.jsx > ...
1   import React from 'react';
2
    Codeium: Refactor | Explain | Generate JSDoc
3   const Image = ({ image, handleRemove }) => {
4     return (
5       <div className="image">
6         <button className="remove-button" onClick={() =>
          handleRemove(image)}>
7           Remove
8         </button>
9         <img src={image} alt="Snapshot" />
10      </div>
11    );
12  };
13
14  export default Image;
15
16
```

Description: The Image.jsx component represents an individual image in the gallery. It displays the image and provides a button to remove it.

Analysis: This component showcases the presentation of individual images and the option to remove them from the gallery.

ImageUploadValidation.jsx

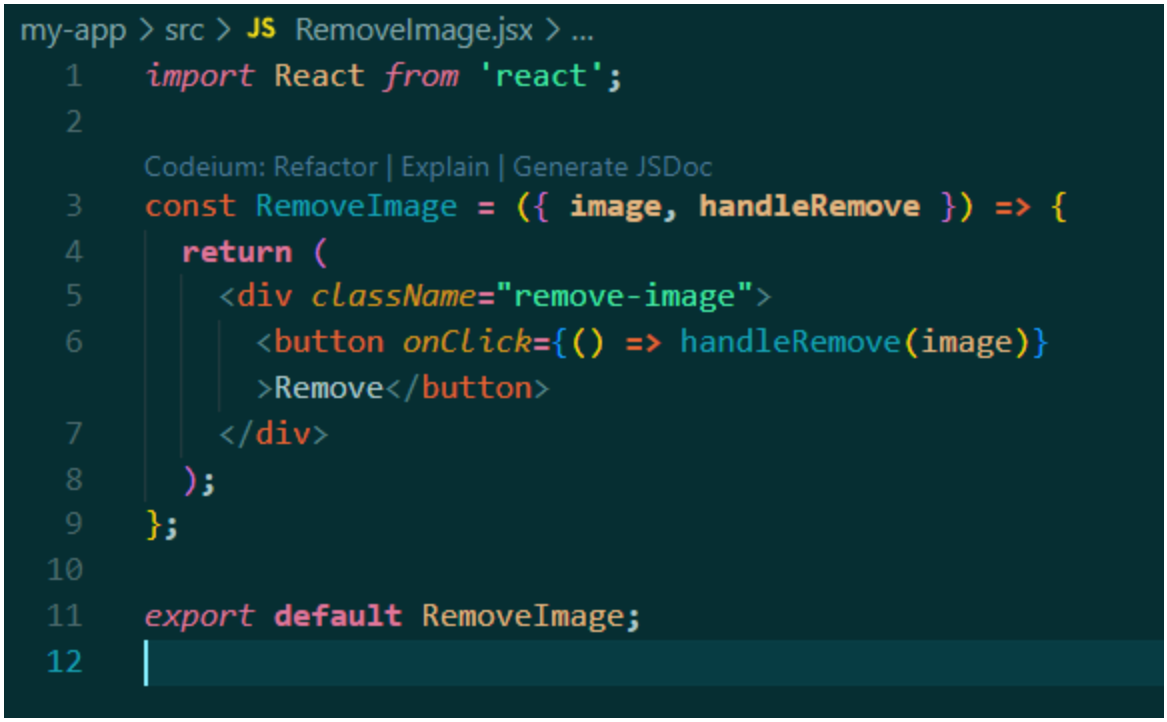
my-app > src > JS ImageUploadValidation.jsx > ...

```jsx
import React, { useState } from 'react';

Codeium: Refactor | Explain | Generate JSDoc
const ImageUploadValidation = ({ handleAdd }) => {
  const [file, setFile] = useState(null);
  const [error, setError] = useState('');

  Codeium: Refactor | Explain | Generate JSDoc
  const handleFileChange = (e) => {
    const selectedFile = e.target.files[0];

    // Check if the selected file is an image
    if (selectedFile && selectedFile.type.startsWith
    ('image/')) {
      setFile(URL.createObjectURL(selectedFile));
      setError('');
    } else {
      setFile(null);
      setError('Please select a valid image file.');
    }
  };

  Codeium: Refactor | Explain | Generate JSDoc
  const handleSubmit = (e) => {
    e.preventDefault();
    if (file) {
      handleAdd(file);
    }
  };

  return (
    <div className="add-image">
      <form onSubmit={handleSubmit}>
        <input
          type="file"
          accept="image/*"
          onChange={handleFileChange}
```

Description: This component handles the validation of uploaded images. It ensures that only valid image files are added to the gallery.

Analysis: The ImageUploadValidation component demonstrates input validation and user-friendly error handling.

RemoveImage.jsx



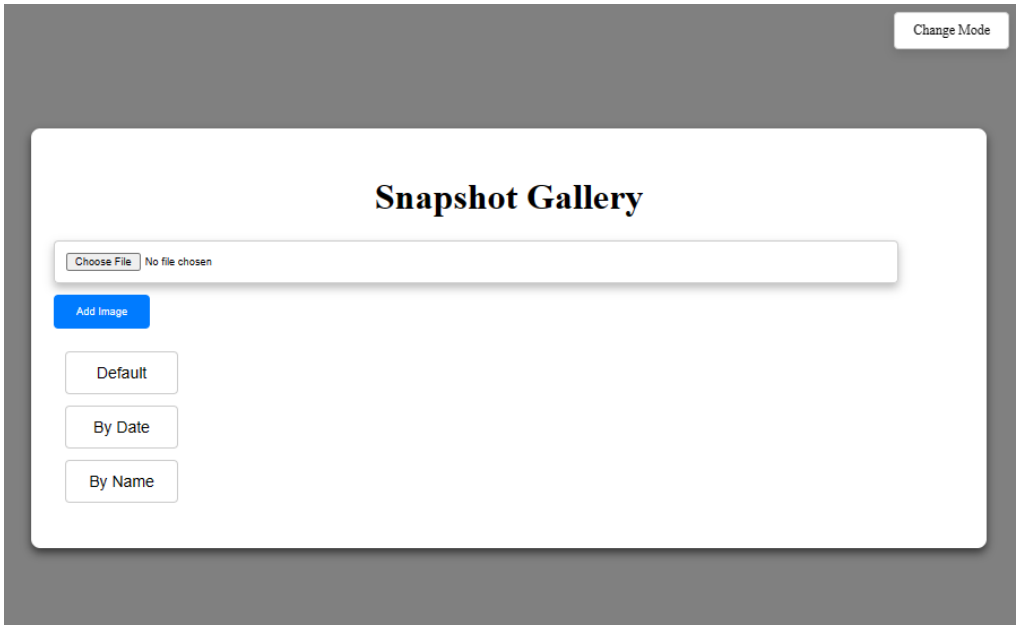
Description: The `RemoveImage.jsx` component plays a crucial role in the application by providing users with the ability to remove specific images from the gallery. It renders a "Remove" button associated with each image, allowing for straightforward deletion.

Analysis: This component enhances user control and customization within the gallery. By incorporating a targeted removal feature, it empowers users to manage their content effectively. The component's intuitive design aligns with the overall user-centric approach of the application, offering a seamless experience for modifying the gallery's content.

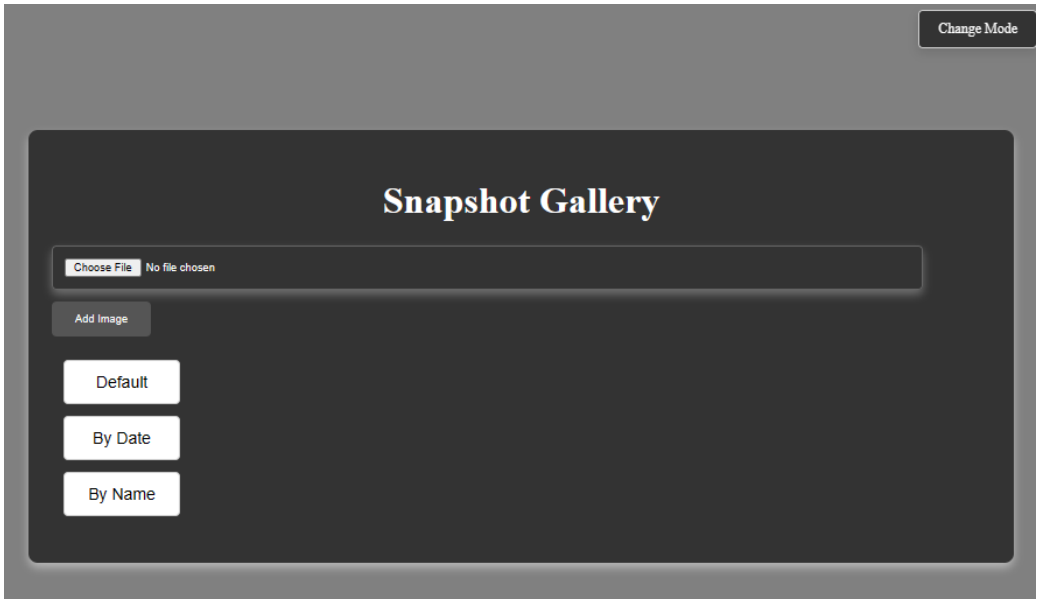
LoadingState.jsx

Description: LoadingState.jsx` is a vital component that offers visual feedback to users during the image upload process. It displays a message indicating "Uploading image..." to notify users of the ongoing operation.
Analysis: This component serves to improve user experience by providing real-time feedback on the upload progress. It effectively manages user expectations, preventing confusion or frustration during potentially time-consuming operations. By simulating a delay, it mimics real-world scenarios, ensuring users are aware of the ongoing process.

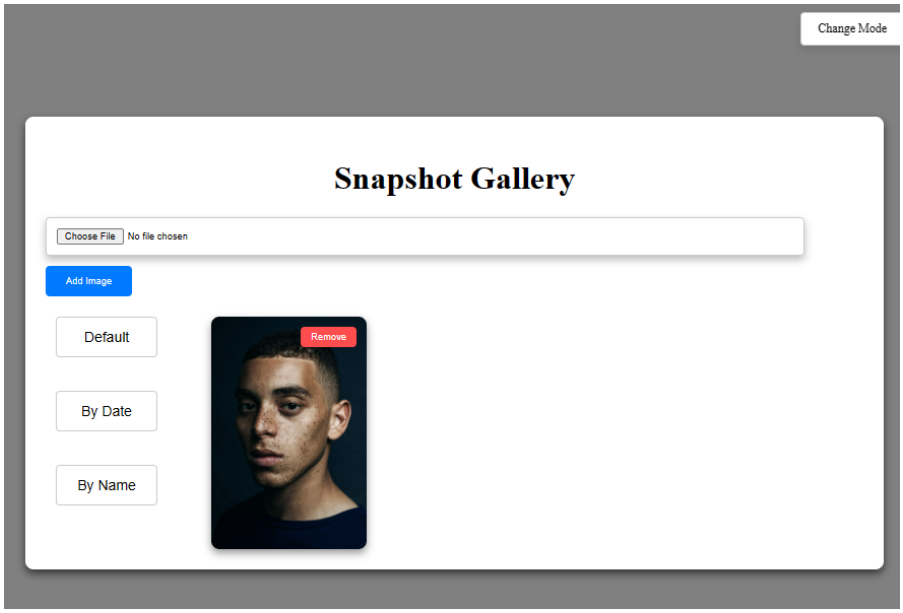
Project Screenshots



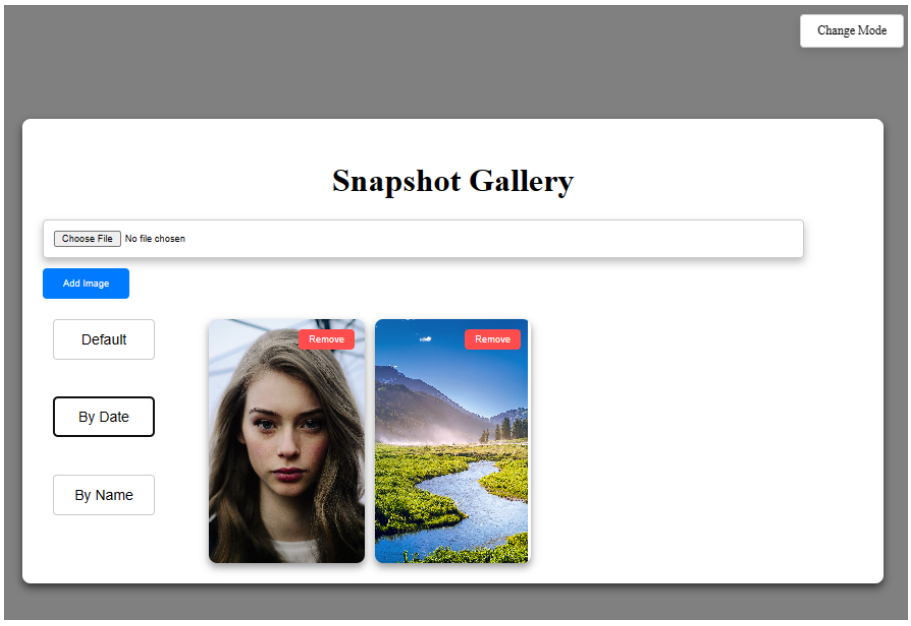
Home page



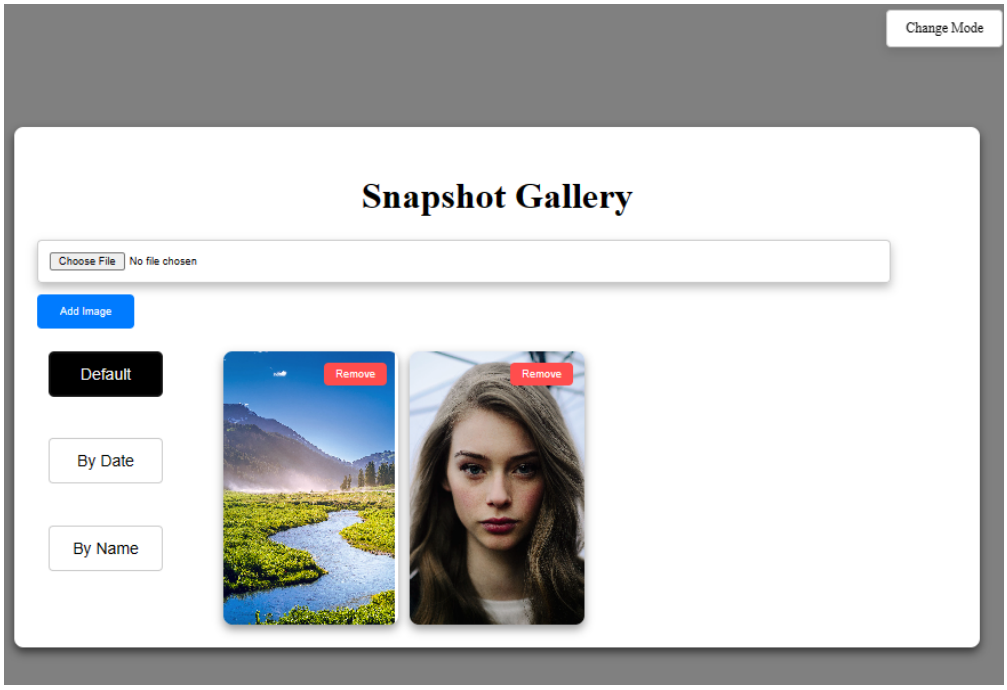
Home page with dark mode



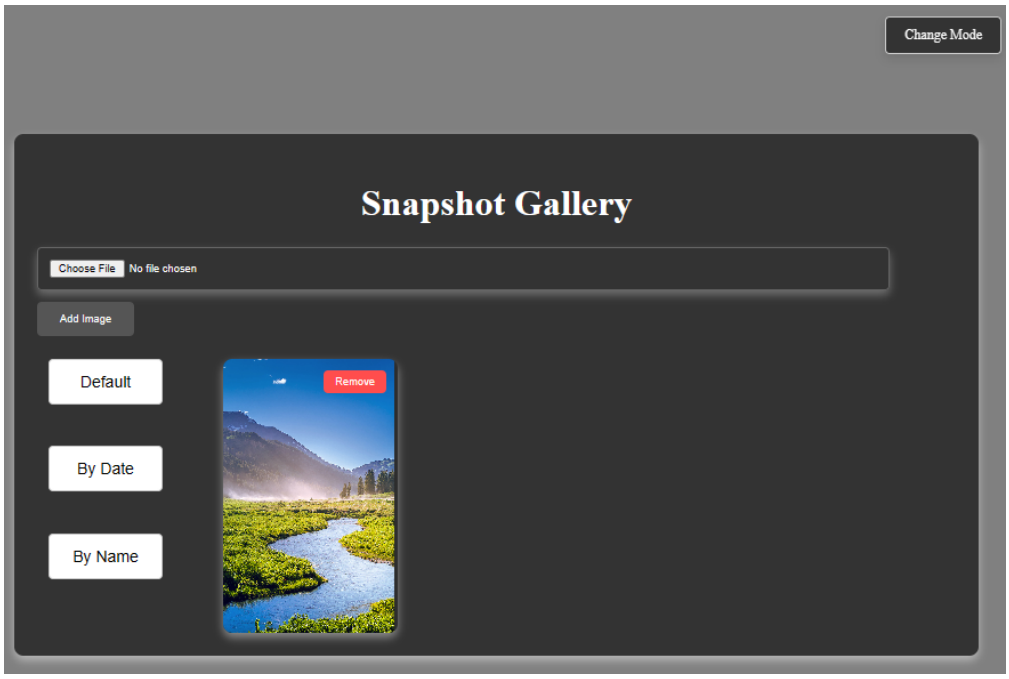
Adding an image



Adding 2 or more than 2 images and sorting with “By Date”



Sorting with “Default”



Gallery view with dark mode while images are there