JavaScript: how to get data URLs from images

In web development, a common feature is to allow users to upload images to forms, which can then be viewed or processed before being sent to a server. JavaScript offers various tools to handle these types of operations, including the `FileReader` API and the `Promise` concept. In this article, we will explore how to use `FileReader` in conjunction with `Promise` to obtain a data URL from an uploaded image, thus allowing it to be viewed or processed on the client side.

The goal is to read a user-selected image file (for example, through a file-type input field) and convert it to a data URL that can be displayed in a `<img>` or used for other processing. This process involves creating an instance of `FileReader` and using the `readAsDataURL` method, which reads the file and encodes it in base64, generating a URL representing the data of the image.

Here's how you can do it, incorporating `Promise` to handle asynchrony:

```
function getFileDataUrl(file) {
   return new Promise((resolve, reject) => {
     // Create a FileReader instance
     const reader = new FileReader();

     // Manages file loading
     reader.onload = () => resolve(reader.result);

     // Handle any errors
     reader.onerror = error => reject(error);
```

```
      // Read the file as a data URL
      reader.readAsDataURL(file);
    });
}
```

After defining the `getFileDataUrl` function, here's how it can be used in practice:

```
// Takes the reference to the input element of type
file
const fileInput = document.querySelector('.file');

fileInput.addEventListener('change', event => {
   // Gets the first selected file
   const file = event.target.files[0];

   if (file) {
     getFileDataUrl(file).then(url => {
       // Use the data URL, for example displaying it
in an img element
       document.querySelector('img').src = url;
     }).catch(error => {
       console.error("Error reading the file:",
error);
     });
   }
});
```

This code adds a handler for the `change` event of the file input element. When a file is selected, the `getFileDataUrl` function is called with the file

as the argument. The returned promise is then resolved with the URL of the image data, which can be used to set the `src` attribute of a `<img>` element, allowing it to visualization.

Conclusion

Using `FileReader` and `Promise` together provides a powerful and efficient way to handle loading and reading files in JavaScript asynchronously. This approach not only improves the maintainability of the code but also opens the door to a wide range of client-side data manipulation possibilities, improving interactivity and user experience in modern web applications.