Sprint3CodeSummary

Bryson Brandon

11/24/23

CSCI-4250-002

Kinser

```
<div class="div-center">
    <button class="btn-primary" onclick="requestCameraAccess()">Scan QR Code</button>

    <div id="cameraModal" class="camera-modal">
        <div class="camera-modal-content">
            <span class="close" onclick="closeCameraModal()">&times;</span>
            <video id="camera" autoplay></video>
        </div>
    </div>
</div>
```

HTML Structure and Button Trigger

I added a new HTML structure to the project, introducing a container for QR code scanning functionality. Within this container, a button with the class "btn-primary" triggers the execution of the "requestCameraAccess" JavaScript function when clicked. This function is responsible for requesting camera access and displaying the camera stream in a modal. The modal, initially hidden, includes a video element for streaming the camera feed and a close button.

```
function requestCameraAccess() {
    navigator.mediaDevices.getUserMedia({ video: true })
        .then((stream) => {
            const video = document.getElementById('camera');
            video.srcObject = stream;
            document.getElementById('cameraModal').style.display = 'block';
            video.play();
            startQRCodeScanner(video);
        })
        .catch((error) => {
            console.error('Error accessing camera:', error);
        });
}

    /**
    * Starts the QR code scanner by continuously capturing frames from the camera stream.
    * Invokes the handleQRCodeDetected function when a QR code is successfully detected.
    * param {HTMLVideoElement} video - The video element displaying the camera stream.
    */
    function startQRCodeScanner(video) {
        const canvas = document.createElement('canvas');
        const context = canvas.getContext('2d');

        const scanFrame = () => {
            context.drawImage(video, 0, 0, canvas.width, canvas.height);
            const imageData = context.getImageData(0, 0, canvas.width, canvas.height);
            const code = jsQR(imageData.data, imageData.width, imageData.height);
            if (code) {
                handleQRCodeDetected(code.data);
            } else {
                requestAnimationFrame(scanFrame);
            }
        };

        video.addEventListener('loadeddata', () => {
            canvas.width = video.videoWidth;
            canvas.height = video.videoHeight;
            scanFrame();
        });
    }

    /**
    * Handles the detected QR code data.
    * Logs the QR code data to the console and closes the camera modal.
    * param {string} qrCodeData - The data extracted from the detected QR code.
    */
    function handleQRCodeDetected(qrCodeData) {
        // Handle the detected QR code data
        console.log('Detected QR Code:', qrCodeData);
        closeCameraModal();
    }

    /**
    * Closes the camera modal, pauses the video stream, and stops camera tracks.
    */
    function closeCameraModal() {
        const video = document.getElementById('camera');
        const modal = document.getElementById('cameraModal');
        video.pause();
        video.srcObject.getTracks().forEach(track => track.stop());
        modal.style.display = 'none';
    }
```

Camera Access and QR Code Scanning

In this commit, I implemented the "requestCameraAccess" function, utilizing the navigator.mediaDevices.getUserMedia API to request access to the user's camera. Upon successfully obtaining access, the camera modal is displayed, and the camera stream is played within the video element. Additionally, I integrated the "startQRCodeScanner" function, which continuously captures frames from the camera stream and invokes the "handleQRCodeDetected" function when a QR code is successfully detected.

QR Code Scanner Logic and Data Handling

This commit introduces the core logic for the QR code scanning functionality. The "startQRCodeScanner" function creates a canvas, captures frames from the camera stream, and utilizes the jsQR library to detect QR codes. When a QR code is detected, the "handleQRCodeDetected" function is invoked, logging the QR code data to the console and closing the camera modal. This functionality currently is able to detect a QR code and log the information contained in it into the console of the browser. The user is then routed to the link contained in the QR code.The "handleQRCodeDetected" function also closes the camera modal, providing a seamless user experience.

Closing Camera Modal and Cleanup

In this commit, I implemented the "closeCameraModal" function to handle the closing of the camera modal. This function pauses the video stream, stops camera tracks, and hides the modal. This ensures proper cleanup when the user finishes scanning QR codes.

```
.camera-modal {
    display: none;
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: rgba(0, 0, 0, 0.7);
}

.camera-model-content{
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    background-color: #fefefe;
    padding: 20px;
    border: 1px solid #888;
}
.close {
    color: #aaa;
    float: right;
    font-size: 28px;
    font-weight: bold;
}
```

CSS Styling for Camera Modal

This commit introduces three CSS styles to define the appearance of the camera modal. The "camera-modal" class sets the modal to initially be hidden and covers the entire screen with a semi-transparent background. The "camera-modal-content" class styles the modal content, centering it on the screen. Finally, the "close" class styles the close button, positioning it in the top-right corner of the modal with a specific appearance. These styles enhance the visual presentation of the QR code scanning feature.

Testing

For testing the camera capabilities, I ran a few basic tests, those being to check that the website was asking me for permission, and logging that in the console. I also tested this feature on my laptop with a webcam, which displayed, and my desktop with no webcam, which automatically detected OBS as my video output device.