Przechwytywanie obrazu od użytkownika

Większość przeglądarek może uzyskać dostęp do aparatu użytkownika.

Wagi matowe

Twitter (https://twitter.com/wibblymat)

Paul Kinlan

Twitter (https://twitter.com/paul_kinlan) GitHub (https://github.com/PaulKinlan) Glitch (https://glitch.com/@PaulKinlan)

Mastodon (https://status.kinlan.me/@paul) strona główna (https://paul.kinlan.me/)

Wiele przeglądarek ma teraz możliwość korzystania z wejścia wideo i audio od użytkownika. W zależności od przeglądarki może on jednak być w pełni dynamiczny i wbudowany lub zostać przekazany do innej aplikacji na urządzeniu użytkownika. Co więcej, nie każde urządzenie ma aparat. Jak więc stworzyć środowisko korzystające z obrazu wygenerowanego przez użytkownika i dopasowane do wszystkich celów?

Zacznij prosto i stopniowo

Jeśli chcesz stopniowo zwiększać komfort korzystania z usługi, zacznij od rozwiązania, które sprawdza się wszędzie. Najprostszym sposobem jest poproszenie użytkownika o przesłanie wcześniej nagranego pliku.

Poproś o adres URL

To najlepsza obsługiwana opcja, ale najmniej satysfakcjonująca. Poproś użytkownika o podanie adresu URL i użyj go. Ta opcja działa wszędzie, jeśli tylko obraz jest wyświetlany. Utwórz element **`img`**, ustaw **`src`** i gotowe.

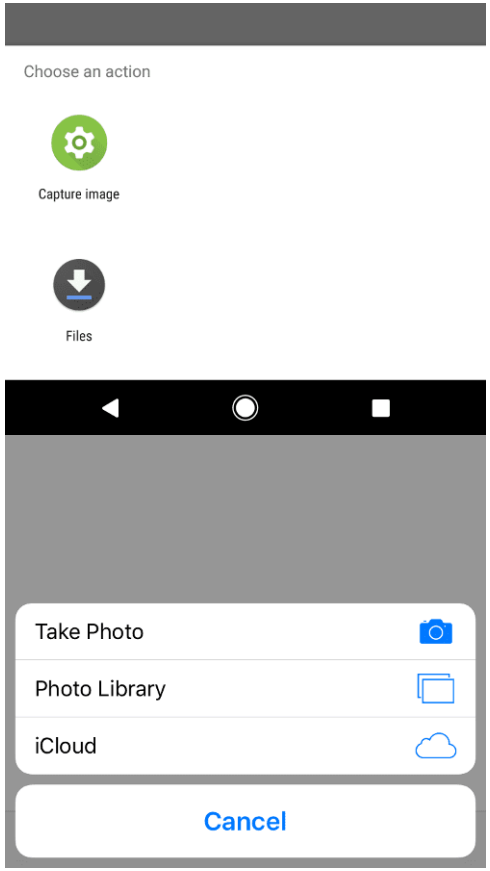
Jeśli jednak chcesz manipulować obrazem w jakikolwiek sposób, sprawa jest nieco bardziej złożona. CORS (https://developer.mozilla.org/docs/Web/HTTP/Access_control_CORS) uniemożliwia dostęp do rzeczywistych pikseli, chyba że serwer ustawi odpowiednie nagłówki i oznaczysz obraz jako crossorigin (https://developer.mozilla.org/docs/Web/HTML/CORS_enabled_image). Jedynym praktycznym sposobem na uniknięcie tego problemu jest użycie serwera proxy.

Dane wejściowe pliku

Możesz też użyć prostego elementu wejściowego pliku, w tym filtra **`accept`**, który wskazuje, że potrzebujesz tylko plików graficznych.

```
<input type="file" accept="image/*" />
```

Ta metoda działa na wszystkich platformach. Na komputerach wyświetla się prośba o przesłanie pliku graficznego z systemu plików. W Chrome i Safari na urządzeniach z iOS i Androidem użytkownik może wybrać aplikację, której chce użyć do zrobienia zdjęcia. Może też zrobić zdjęcie bezpośrednio aparatem lub wybrać istniejący plik obrazu.



Dane można dołączyć do elementu **`<form>`** lub zmodyfikować za pomocą JavaScriptu, wykrywając zdarzenie **`onchange`** w elemencie wejściowym, a następnie odczytując właściwość **`files`** zdarzenia **`target`**.

```
<input type="file" accept="image/*" id="file-input" />
<script>
  const fileInput = document.getElementById('file-input');
```

```
  fileInput.addEventListener('change', (e) =>
    doSomethingWithFiles(e.target.files),
  );
</script>
```

Właściwość **`files`** to obiekt **`FileList`**, o którym wspomnię później.

Do elementu możesz też opcjonalnie dodać atrybut **`capture`**, który informuje przeglądarkę, że wolisz pobierać obraz z aparatu.

```
<input type="file" accept="image/*" capture />
<input type="file" accept="image/*" capture="user" />
<input type="file" accept="image/*" capture="environment" />
```

Dodanie atrybutu **`capture`** bez wartości pozwala przeglądarce zdecydować, której kamery użyć, a wartości "**`user`**" i "**`environment`**" informują przeglądarkę, czy powinien preferować aparat przedni i tylny.

Atrybut **`capture`** działa na urządzeniach z Androidem i iOS, ale na komputerach jest ignorowany. Pamiętaj jednak, że na Androidzie oznacza to, że użytkownik nie będzie już mógł wybrać istniejącego zdjęcia. Aplikacja aparatu systemowego zostanie uruchomiona bezpośrednio.

Przeciągnij i upuść

Jeśli dodajesz już możliwość przesyłania pliku, możesz skorzystać z kilku prostych sposobów na urozmaicenie przeglądania.

Pierwszym z nich jest dodanie do strony miejsca docelowego, który umożliwia użytkownikowi przeciąganie pliku z komputera lub innej aplikacji.

```
<div id="target">You can drag an image file here</div>
<script>
  const target = document.getElementById('target');

  target.addEventListener('drop', (e) => {
    e.stopPropagation();
    e.preventDefault();

    doSomethingWithFiles(e.dataTransfer.files);
  });

  target.addEventListener('dragover', (e) => {
    e.stopPropagation();
    e.preventDefault();

    e.dataTransfer.dropEffect = 'copy';
  });
</script>
```

Podobnie jak w przypadku danych wejściowych, obiekt **`FileList`** możesz uzyskać z właściwości **`dataTransfer.files`** zdarzenia **`drop`**.

Moduł obsługi zdarzeń **`dragover`** za pomocą właściwości **`dropEffect`** (https://developer.mozilla.org/docs/Web/API/DataTransfer/dropEffect) informuje użytkownika, co się stanie, gdy usunie plik.

Funkcja przeciągania i upuszczania jest stosowana od dawna i jest dobrze obsługiwana w najpopularniejszych przeglądarkach.

Wklej ze schowka

Ostatnim sposobem na uzyskanie pliku graficznego jest użycie schowka. Kod jest bardzo prosty, ale jej prawidłowe działanie jest nieco trudniejsze.

```
<textarea id="target">Paste an image here</textarea>
<script>
  const target = document.getElementById('target');

  target.addEventListener('paste', (e) => {
    e.preventDefault();
    doSomethingWithFiles(e.clipboardData.files);
  });
</script>
```

(`e.clipboardData.files` to kolejny obiekt `FileList`).

W przypadku interfejsu API schowka problem polega na tym, że aby zapewnić pełną obsługę różnych przeglądarek, element docelowy musi być zarówno zaznaczony, jak i edytowalny. Zarówno `<textarea>`, jak i `<input type="text">` pasują tutaj, podobnie jak elementy z atrybutem `contenteditable`. Są one jednak oczywiście przeznaczone do edytowania tekstu.

Jej płynne działanie może być trudne, jeśli nie chcesz, by użytkownik mógł wpisywać tekst. Takie sztuczki jak ukrywanie danych wejściowych, które są wybierane po kliknięciu innego elementu, mogą utrudniać utrzymanie ułatwień dostępu.

Obsługa obiektu FileList

Ponieważ większość powyższych metod generuje **`FileList`**, muszę dokładniej wyjaśnić, co to jest.

`FileList` jest podobny do **`Array`**. Ma klucze liczbowe i właściwość **`length`**, ale *w rzeczywistości* nie jest tablicą. Nie ma metod tablicy, takich jak **`forEach()`** czy **`pop()`**, i nie jest wykonywana iteracyjna. Prawdziwą tablicę można oczywiście uzyskać, korzystając z metody **`Array.from(fileList)`**.

Pozycje **`FileList`** mają obiekty **`File`**. Są one dokładnie takie same jak obiekty **`Blob`** z tą różnicą, że mają dodatkowe właściwości **`name`** i **`lastModified`** tylko do odczytu.

```
<img id="output" />
<script>
  const output = document.getElementById('output');

  function doSomethingWithFiles(fileList) {
    let file = null;

    for (let i = 0; i < fileList.length; i++) {
      if (fileList[i].type.match(/^image\//)) {
        file = fileList[i];
        break;
      }
    }

    if (file !== null) {
      output.src = URL.createObjectURL(file);
    }
  }
</script>
```

W tym przykładzie znajdujemy pierwszy plik z typem MIME obrazu, ale może on też obsłużyć wiele obrazów wybieranych, wklejanych i usuwanych jednocześnie.

Gdy masz już dostęp do pliku, możesz z nim robić wszystko, co tylko chcesz. Możesz na przykład:

- Narysuj go w elemencie `<canvas>`, aby móc nim manipulować
- Pobierz na urządzenie użytkownika
- Prześlij na serwer za pomocą `fetch()`

Interaktywny dostęp do aparatu

Masz już wszystkie podstawy, więc czas na ich stopniowe ulepszanie.

Nowoczesne przeglądarki mają bezpośredni dostęp do kamer, dzięki czemu mogą tworzyć interfejsy, które są w pełni zintegrowane ze stroną internetową, a użytkownik nie musi nigdy opuszczać przeglądarki.

Uzyskiwanie dostępu do aparatu

Dostęp do kamery i mikrofonu możesz uzyskać bezpośrednio za pomocą interfejsu API w specyfikacji WebRTC o nazwie `getUserMedia()`. Spowoduje to wyświetlenie prośby dla użytkownika o dostęp do podłączonych mikrofonów i kamer.

Obsługa języka `getUserMedia()` jest dość wysoka, ale nie jest jeszcze dostępna wszędzie. Nie jest ona dostępna w przeglądarce Safari w wersji 10 i starszych, która w momencie tworzenia tego tekstu jest wciąż stabilną wersją. Firma Apple ogłosiła jednak (https://webkit.org/blog/7726/announcing-webrtc-and-media-capture/) jednak, że ta wersja będzie dostępna w przeglądarce Safari 11.

Można je jednak bardzo łatwo wykryć.

```
const supported = 'mediaDevices' in navigator;
```

Ostrzeżenie: bezpośredni dostęp do aparatu to zaawansowana funkcja. Wymaga zgody użytkownika, a witryna MUSI prowadzić do bezpiecznego źródła (HTTPS).

Wywołując funkcję **`getUserMedia()`**, musisz przekazać obiekt opisujący rodzaj multimediów, jakich potrzebujesz. Takie ograniczenia są nazywane ograniczeniami. Istnieje kilka możliwych ograniczeń. Obejmują one wiele kwestii, takich jak preferowany aparat przedni czy tylny, wybór dźwięku oraz preferowana rozdzielczość transmisji.

Aby jednak pobierać dane z kamery, potrzebujesz tylko jednego ograniczenia, którym jest **`video: true`**.

Jeśli operacja się uda, interfejs API zwróci obiekt **`MediaStream`** zawierający dane z kamery. Możesz go następnie dołączyć do elementu **`<video>`** i odtworzyć, aby wyświetlić podgląd w czasie rzeczywistym, lub dołączyć go do obiektu **`<canvas>`**, aby uzyskać zrzut.

```
<video id="player" controls autoplay></video>
<script>
  const player = document.getElementById('player');

  const constraints = {
    video: true,
  };
```

```
  navigator.mediaDevices.getUserMedia(constraints).then((stream) => {
    player.srcObject = stream;
  });
</script>
```

To nie jest zbyt przydatne. Wystarczy pobrać dane filmu i odtworzyć je. Jeśli chcesz uzyskać obraz, musisz wykonać kilka dodatkowych czynności.

Zrób zrzut ekranu

Najlepszą obsługiwaną opcją w przypadku obrazu jest narysowanie klatki z filmu na obraz na płótnie.

W przeciwieństwie do Web Audio API nie ma specjalnego interfejsu API do przetwarzania strumieni wideo w internecie, więc aby wykonać zrzut ekranu z kamery użytkownika, musisz zastosować drobne zmiany hakerskie.

Proces przebiega w ten sposób:

1. Utwórz obiekt na płótnie, który będzie utrzymywać kadr z aparatu
2. Uzyskiwanie dostępu do transmisji z kamery
3. Dołącz do elementu wideo
4. Jeśli chcesz uchwycić dokładną klatkę, dodaj dane z elementu wideo do obiektu canvas za pomocą funkcji `drawImage()`.

```
<video id="player" controls autoplay></video>
<button id="capture">Capture</button>
```

```
<canvas id="canvas" width="320" height="240"></canvas>
<script>
  const player = document.getElementById('player');
  const canvas = document.getElementById('canvas');
  const context = canvas.getContext('2d');
  const captureButton = document.getElementById('capture');

  const constraints = {
    video: true,
  };

  captureButton.addEventListener('click', () => {
    // Draw the video frame to the canvas.
    context.drawImage(player, 0, 0, canvas.width, canvas.height);
  });

  // Attach the video stream to the video element and autoplay.
  navigator.mediaDevices.getUserMedia(constraints).then((stream) => {
    player.srcObject = stream;
  });
</script>
```

Dane z kamery zapisane w obszarze roboczym można wykorzystać na wiele różnych sposobów. Możesz:

- Prześlij go bezpośrednio na serwer.
- Przechowuj lokalnie
- Zastosuj dziwne efekty do zdjęcia

Wskazówki

Zatrzymuj przesyłanie strumieniowe z kamery, gdy nie są potrzebne

Zaleca się przerwanie używania kamery, gdy nie jest już potrzebna. W ten sposób nie tylko oszczędzasz baterię i moc obliczeniową, ale też zwiększasz zaufanie użytkowników do swojej aplikacji.

Aby wyłączyć dostęp do kamery, możesz po prostu wywołać funkcję **`stop()`** w każdej ścieżce wideo w przypadku strumienia zwróconego przez użytkownika **`getUserMedia()`**.

```
<video id="player" controls autoplay></video>
<button id="capture">Capture</button>
<canvas id="canvas" width="320" height="240"></canvas>
<script>
  const player = document.getElementById('player');
  const canvas = document.getElementById('canvas');
  const context = canvas.getContext('2d');
  const captureButton = document.getElementById('capture');

  const constraints = {
    video: true,
  };

  captureButton.addEventListener('click', () => {
    context.drawImage(player, 0, 0, canvas.width, canvas.height);

    // Stop all video streams.
    player.srcObject.getVideoTracks().forEach(track => track.stop());
  });
```

```
  navigator.mediaDevices.getUserMedia(constraints).then((stream) => {
    // Attach the video stream to the video element and autoplay.
    player.srcObject = stream;
  });
</script>
```

Poproś o zgodę na odpowiedzialne korzystanie z kamery

Jeśli użytkownik nie przyznał Twojej witrynie wcześniej dostępu do kamery, natychmiast po wywołaniu metody **`getUserMedia()`** przeglądarka poprosi użytkownika o udzielenie uprawnień do korzystania z kamery.

Użytkownicy nienawidzą próśb o dostęp do zaawansowanych urządzeń i często blokują je lub zignorują je, jeśli nie rozumieją kontekstu, dla którego został utworzony. Najlepiej prosić o dostęp do kamery tylko wtedy, gdy jest to konieczne. Gdy użytkownik przyzna dostęp, nie będzie więcej o to pytany. Jeśli jednak użytkownik odrzuci dostęp, nie będzie można go odzyskać, chyba że ręcznie zmieni on ustawienia uprawnień do korzystania z aparatu.

Ostrzeżenie: jeśli poprosisz o dostęp do aparatu podczas wczytywania strony, większość użytkowników go odrzuci.

Zgodność

Więcej informacji o implementacji przeglądarek na urządzeniach mobilnych i komputerach:

- srcObject (https://www.chromestatus.com/feature/5989005896187904)

- navigator.mediaDevices.getUserMedia() (https://www.chromestatus.com/features/5755699816562688)

Zalecamy też używanie podkładki adapter.js (https://github.com/webrtc/adapter) do ochrony aplikacji przed zmianami specyfikacji WebRTC i różnicami prefiksów.

Prześlij opinię

Ostatnia aktualizacja: 2016-08-23 UTC.