

Tesi - Payment Request API

Daniele Rigon - 857319

24 luglio 2018

Indice

1	Overview Payment Request	2
1.1	Concetti e utilizzo della richiesta di pagamento	2
1.2	Vantaggi nell'utilizzo di questa API	2
1.3	Come funziona	2
1.4	Rischi	3
1.5	Sicurezza API	3
1.6	Uso API	4
1.6.1	Ruolo dell'utente	4
1.6.2	Ruolo del commerciante	4
1.6.3	Ruolo del fornitore app di pagamento	4
1.6.4	Ruolo del browser	4
1.6.5	Metodi di pagamento	4
1.6.6	App di pagamento	5
1.6.7	Differenze tra metodo di pagamento e app di pagamento	6
2	Specifiche	6
2.1	Metodi	6
2.1.1	MethodData	7
2.1.2	Details	7
2.1.3	Opzioni di spedizione	8
2.1.4	Modifiche condizionali alla richiesta di pagamento	8
2.1.5	Options	9
2.1.6	PaymentRequest	9
2.1.7	Gestione degli eventi e aggiornamento della richiesta di pagamento	10
2.1.8	Segnalazione di errori a grana fine	11
2.1.9	Postare la risposta di pagamento su un server	11
3	Codice d'esempio per pagina demo	12
3.0.1	Costruttore	12
3.0.2	Visualizzazione dell'interfaccia utente, elaborazione del pagamento e visualizzazione dei risultati	13
3.0.3	Ascoltando gli eventi	15
3.0.4	Rilevamento di funzionalità	15
4	Compatibilità web	18
5	Conclusioni	19

1 Overview Payment Request

Fare acquisti sul web, in particolare sui dispositivi mobili, può essere un'esperienza frustrante, in quanto ogni sito Web ha il proprio sistema e la maggior parte dei siti richiede agli utenti di digitare manualmente le stesse informazioni (di contatto, credenziali di pagamento, indirizzi) più e più volte, e questa ridondanza può portare alla perdita di clienti. Allo stesso modo creare e gestire pagine che supportano diversi metodi di pagamento può risultare difficile, e può richiedere molto tempo agli sviluppatori. La PaymentRequest API consente ai commercianti di creare esperienze di pagamento semplificate. Piuttosto che ridigitare le stesse informazioni più volte nel Web, gli utenti possono memorizzare e riutilizzare le informazioni e completare più rapidamente e accuratamente le transazioni online.

1.1 Concetti e utilizzo della richiesta di pagamento

Molti problemi legati all'abbandono degli acquisti online possono essere ricondotti ai moduli di checkout, difficili da usare, lenti da caricare e aggiornare e richiedono più passaggi da completare. La PaymentRequest API è un sistema che ha lo scopo di eliminare i moduli di checkout, migliorando il flusso di lavoro degli utenti durante il processo di acquisto, offrendo un'esperienza utente più coerente e consentendo ai commercianti di sfruttare facilmente diversi metodi di pagamento.

1.2 Vantaggi nell'utilizzo di questa API

- Esperienza di acquisto rapida: gli utenti immettono i propri dati una volta nel browser e sono pronti a pagare beni e servizi sul Web; una volta inseriti i dati non è più necessario compilare ripetutamente gli stessi dettagli su siti diversi;
- Esperienza coerente su ogni sito (che supporta l'API): poiché la pagina di pagamento è controllata dal browser si può personalizzare l'esperienza utente, ad esempio includendo la localizzazione per impostare automaticamente la lingua preferita dell'utente, ecc;
- Gestione delle credenziali: gli utenti possono gestire le loro carte di credito e gli indirizzi di spedizione direttamente nel browser. Un browser può anche sincronizzare queste "credenziali" tra dispositivi, rendendo più semplice per gli utenti passare dal desktop al cellulare e viceversa quando si acquistano oggetti;
- Gestione coerente degli errori: il browser può controllare la validità dei numeri delle carte e può comunicare all'utente se una carta è scaduta o sta per scadere, può suggerire automaticamente quale carta utilizzare in base ai modelli di utilizzo passati o alle restrizioni del commerciante, o consentire all'utente di dire quale sia la carta predefinita/preferita.
- Esperienza utente migliorata: meno tipizzazione, coerenza tra i siti Web, coerenza tra browser e sistemi operativi e nuove funzionalità del browser per semplificare il checkout, ecc;
- Miglioramento della sicurezza: la PaymentRequest API ha il potenziale per ridurre le opportunità di frode e può facilitare l'adozione di metodi di pagamento più sicuri;
- Responsabilità inferiore: in passato, per creare un'esperienza utente semplificata, i commercianti dovevano memorizzare le credenziali di pagamento degli utenti. Questo non è più necessario, il che può aiutare a ridurre la responsabilità del commerciante nei confronti del cliente;

1.3 Come funziona

L'API di richiesta di pagamento consente a un utente di completare una transazione più facilmente riutilizzando le informazioni memorizzate nel browser o in app di pagamento di terze parti. Quando l'utente preme un pulsante in una pagina di checkout collegata all'API il commerciante utilizza l'API per richiedere il pagamento. Il commerciante fornisce informazioni su prezzo, valuta e un elenco di metodi di pagamento accettati, e può inoltre richiedere al browser di creare un'interfaccia utente semplificata per raccogliere l'indirizzo di spedizione, le informazioni di contatto e un numero limitato di elementi aggiuntivi dall'utente. Il browser determina quali metodi di pagamento sono supportati dal commerciante tra le varie "app di pagamento" mostrandole all'utente. L'utente seleziona un'app di pagamento con la quale

pagare, la quale può comportare ulteriori interazioni con l'utente (ad esempio per l'autenticazione); al completamento l'app di pagamento restituisce i dati tramite l'API al commerciante.

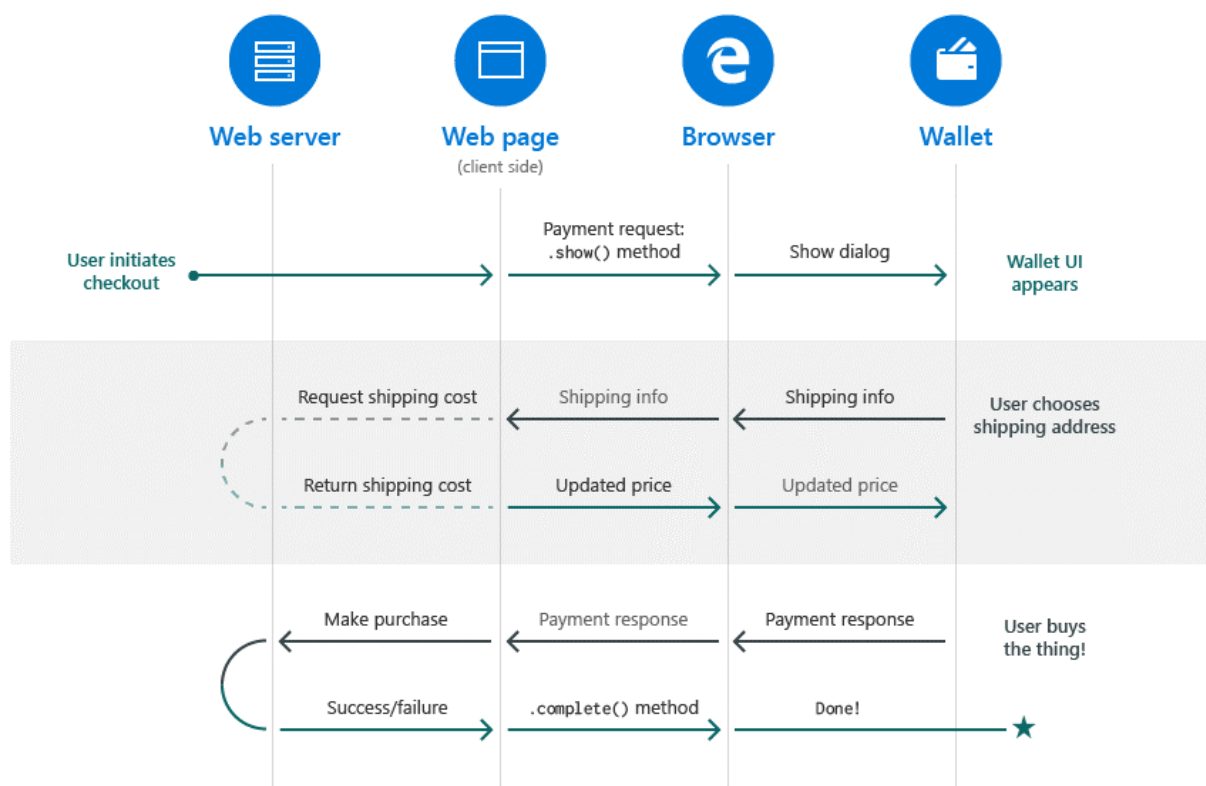


Figura 1: Schema Payment Request API

1.4 Rischi

La PaymentRequest API consente ai commercianti di "consegnare" la raccolta di credenziali di pagamento alle app di pagamento, riducendo così l'onere di creare un'esperienza di pagamento adeguata per l'utente.

1.5 Sicurezza API

La PaymentRequest API aumenta la sicurezza poichè:

- I commercianti possono ottenere un checkout semplificato senza memorizzare le credenziali dell'utente, in quanto lo fa l'API, rendendo i commercianti meno vulnerabili agli attacchi;
- La PaymentRequest API dovrebbe facilitare l'introduzione di metodi di pagamento più sicuri sul Web, come i pagamenti con carta tokenizzata;
- I proprietari dei metodi di pagamento disporranno di meccanismi standard per autorizzare software specifici a implementare il loro metodo di pagamento, che il browser può verificare attraverso una firma digitale.

I browser utilizzano una varietà di meccanismi per archiviare informazioni sensibili dell'utente, e attualmente W3C sta sviluppando nuove tecnologie per aumentarne la sicurezza. Per quanto riguarda la memorizzazione delle informazioni delle app di pagamento in modo sicuro, questo è un dettaglio di implementazione che è diverso per ogni app di pagamento, e tale sicurezza è a carico del provider dell'app di pagamento.

1.6 Uso API

1.6.1 Ruolo dell'utente

Gli utenti beneficiano del riutilizzo delle credenziali inserite nel browser o nelle app di pagamento. Quindi, quando si visita un sito Web che sfrutta la PaymentRequest API gli utenti avranno l'opportunità di sfruttare il riutilizzo semplificato delle credenziali archiviate.

1.6.2 Ruolo del commerciante

L'API influisce sul front end e non sul back-end, ovvero solamente sull'interfaccia dell'esperienza utente; pertanto il commerciante non dovrebbe dover apportare modifiche all'elaborazione back-end di vari metodi di pagamento, questo sarà compito del fornitore della pagina di pagamento, il quale sostituirà i moduli Web con le chiamate alla PaymentRequest API.

È richiesto il gesto dell'utente per attivare l'API di richiesta di pagamento attraverso `PaymentRequest.show()`.

L'API non include l'autenticazione dell'utente, ma esistono due tipi di autenticazione utente che esulano dall'ambito delle specifiche della PaymentRequest API:

- Autenticazione del commerciante dell'utente: l'utente accede al proprio account con il commerciante;
- Autenticazione delle app di pagamento dell'utente: fornendo un codice CVV, tramite nome/password o tramite autenticazione a più fattori.

1.6.3 Ruolo del browser

Il browser svolge diversi ruoli:

- Calcola l'intersezione dei metodi di pagamento accettati dal commerciante e registrati dall'utente;
- Visualizza l'interfaccia utente che consente all'utente di inserire le proprie informazioni;
- Funge da canale per i dati da e verso il commerciante e da e verso l'utente.

1.6.4 Metodi di pagamento

La PaymentRequest API è progettata per funzionare con un gran numero di metodi di pagamento, i quali vengono identificati attraverso due strade:

- I metodi di pagamento definiti da W3C sono identificati come "basic-card" e sono composti da stringhe corte;
- I metodi di pagamento definiti da altre parti sono identificati dagli URL.

1.6.5 App di pagamento

In base alla progettazione, le app di pagamento sono app native per dispositivi mobili e app Web. Il gruppo di lavoro Web Payments sta lavorando su una specifica per come le applicazioni Web si registrano per gestire le richieste di pagamento. Il gruppo di lavoro Web Payments non discute l'integrazione di app di pagamento native su piattaforme proprietarie; che viene gestito dai proprietari di quelle piattaforme. Il gruppo di lavoro Web Payments sta sviluppando una specifica che definisce il modo in cui il browser possa sapere quali app di pagamento ha l'utente. L'obiettivo del gruppo di lavoro Web Payments è stabilire un meccanismo di registrazione unico per i programmi utente su qualsiasi sistema. Per le app di pagamento create con tecnologia nativa (proprietaria), il browser o il sistema operativo sottostante determinano il meccanismo di registrazione e possono variare da sistema a sistema. L'API di richiesta di pagamento, ovvero il browser, determina se un'app di pagamento "corrisponde" a una determinata transazione definendo un algoritmo che considera:

- Se l'app di pagamento supporta uno qualsiasi dei metodi di pagamento accettati dal commerciante. Il commerciante dichiara quali metodi di pagamento supporta attraverso un elenco di identificativi del metodo di pagamento passati attraverso l'API.

- Inoltre, alcuni metodi di pagamento consentono ai commercianti e alle app di pagamento di descrivere più dettagliatamente le condizioni in base alle quali accettano o supportano tali metodi di pagamento. Questa informazione di "capacità" viene anche utilizzata per determinare se un'app di pagamento corrisponde a una determinata transazione.

Al fine di proteggere la privacy degli utenti, i commercianti hanno accesso a informazioni molto limitate sull'ambiente dell'utente. L'API di richiesta di pagamento supporta un meccanismo di query limitato per consentire al commerciante di determinare se l'utente ha "qualcosa" che può essere utilizzato con l'API di richiesta di pagamento, ma che non fornisce informazioni su software specifico. Questa informazione consente ai commercianti di rilevare il supporto per l'API di richiesta di pagamento e quindi di mostrare una pagina di checkout che utilizza l'API, oppure di creare una pagina di fallback se l'utente non è pronto a pagare con qualche app di pagamento. I commercianti possono semplificare le pagine di pagamento in diversi modi, tra cui:

- Non avranno bisogno di usare la pagina di checkout per raccogliere la spedizione e altri dati comuni; il browser può fornirli più rapidamente se l'utente lo ha già archiviato.
- Non avranno bisogno di chiedere agli utenti di scegliere un metodo di pagamento. Invece, possono fornire un singolo pulsante che consente al browser di visualizzare app di pagamento rilevanti.

In che modo l'API di richiesta di pagamento influisce sul flusso dei metodi di pagamento che già supporta? Oggi, il flusso per gli utenti di solito implica qualcosa del genere:

- Scansione di un elenco di metodi di pagamento accettati (non di cui l'utente si preoccupa)
- Scegliere uno e continua
- Per i metodi di pagamento che prevedono il lancio di un'app o la visita a un sito Web, inviare l'utente a quell'app o sito, quindi inviarli di nuovo dopo che il pagamento è stato completato.

L'API di richiesta di pagamento consente un flusso migliorato:

- L'utente preme un pulsante di acquisto singolo (non è richiesta alcuna scansione)
- Il browser visualizza le app di pagamento dell'utente che possono essere utilizzate per la transazione. È probabile che i browser supportino le preferenze dell'utente in modo che, ad esempio, un'app di pagamento venga avviata automaticamente su un determinato sito Web. Ciò semplificherà ulteriormente il checkout.
- Per i metodi di pagamento che prevedono il lancio di un'app o la visita a un sito Web, inviare l'utente a quell'app o sito, quindi inviarli di nuovo dopo che il pagamento è stato completato. Il gruppo di lavoro Web Payments sta discutendo, tuttavia, come possiamo migliorare l'esperienza dell'utente creando un senso più forte di "essere ancora nel contesto commerciale" quando si utilizza un'app o un sito Web da pagare. Più lavoro deve essere fatto su questo argomento.

1.6.6 Differenze tra metodo di pagamento e app di pagamento

Nell'ecosistema dell'API della richiesta di pagamento, disaccoppiamo i dati necessari per effettuare un pagamento dal software utilizzato per raccogliere tali dati e avviare l'elaborazione dei pagamenti. Un metodo di pagamento è caratterizzato dai dati che il commerciante fornisce al pagatore e riceve dal pagatore per essere pagato.

- Esempio: per un metodo di pagamento con carta di base, il commerciante non fornisce dati al pagatore e riceve dal numero di carta del pagatore, dalla data di scadenza, ecc.
- Esempio: per un metodo di pagamento con trasferimento di credito, il commerciante fornisce informazioni sull'account commerciante, ecc. E riceve dalle informazioni del pagatore sulla data di elaborazione, i codici di transazione, ecc.

Un'app di pagamento è il software che l'utente utilizza per pagare. Un'app di pagamento può supportare uno o più metodi di pagamento. Le app di pagamento possono essere implementate utilizzando una varietà di tecnologie, incluse quelle di sistemi operativi nativi o tecnologie Web o di un ibrido. I browser possono anche fungere da app di pagamento, memorizzando le credenziali dell'utente come le informazioni sulla carta. L'API di richiesta di pagamento e l'API di pagamento non risolvono il funzionamento interno delle app di pagamento, ma solo il modo in cui comunicano con il browser. In generale, più app di pagamento possono implementare lo stesso metodo di pagamento. Tuttavia, vi sono casi importanti in cui è disponibile una sola app di pagamento autorizzata a supportare un metodo di pagamento (proprietario). Nei casi in cui più app di pagamento possono servire un determinato metodo di pagamento (ad es. "Carta base"), il commerciante non deve preoccuparsi dell'integrazione software. Il commerciante richiede "dati" attraverso l'API di richiesta di pagamento. Il commerciante (o il suo fornitore di servizi di pagamento) deve gestire tali dati sul back-end, ma non è necessaria alcuna integrazione software aggiuntiva. Allo stesso modo, il disaccoppiamento libera l'utente a scegliere la sua app di pagamento preferita.

2 Specifiche

2.1 Metodi

Per utilizzare l'API, lo sviluppatore deve fornire e tenere traccia di una serie di informazioni chiave. Questi bit di informazioni vengono passati al costruttore `PaymentRequest` come argomenti e successivamente utilizzati per aggiornare la richiesta di pagamento visualizzata all'utente. Vale a dire, questi bit di informazione sono:

- Il `methodData`: Una sequenza di `PaymentMethodData` che rappresenta i metodi di pagamento che il sito supporta;
- I `details`: i dettagli della transazione, come un `PaymentDetailsInit`. Ciò include il costo totale e facoltativamente un elenco di beni o servizi acquistati, beni materiali e opzioni di spedizione. Inoltre, può facoltativamente includere "modificatori" su come vengono effettuati i pagamenti. Ad esempio, "se paghi con una carta di credito di tipo X, incorre in una tassa di elaborazione di 3,00".
- Le `options`: un elenco di `PaymentOptions`, in quanto il sito ha bisogno di consegnare il bene o il servizio (ad esempio, per i beni fisici, il commerciante di solito avrà bisogno di un indirizzo fisico da spedire; per i beni digitali, di solito un'e-mail è sufficiente). Una volta che il `PaymentRequest` è stato costruito, viene presentato all'utente finale tramite il `show()` metodo. I `show()` ritornano una promessa che, una volta che l'utente conferma la richiesta di pagamento, si traduce in una `PaymentResponse`.
- `payment request`: uno sviluppatore crea una `PaymentRequest` per effettuare una richiesta di pagamento. Questo è in genere associato all'utente che avvia un processo di pagamento. La `PaymentRequest` consente agli sviluppatori di scambiare informazioni con l' `user agent` mentre l'utente sta fornendo dati in input. Poiché la visualizzazione simultanea di più interfacce `PaymentRequest` potrebbe confondere l'utente, questa specifica limita lo `user agent` a visualizzarne uno alla volta tramite il metodo `show()`.
- `payment methodData`: un `PaymentMethodData` viene utilizzato per indicare un insieme di metodi di pagamento supportati e qualsiasi dato specifico del metodo di pagamento associato per tali metodi.
- `payment detailsInit`: Il `PaymentDetailsInit` viene utilizzato nella costruzione della richiesta di pagamento.
- `payment options`: Il `PaymentOptions` viene passato al costruttore `PaymentRequest` e fornisce informazioni sulle opzioni desiderate per la richiesta di pagamento.
- `payment response`: un `PaymentResponse` viene restituito quando un utente ha selezionato un metodo di pagamento e approvato una richiesta di pagamento.

2.1.1 MethodData

La sequenza methodData contiene PaymentMethodData contenenti gli identificativi del metodo di pagamento per i metodi di pagamento accettati dal sito Web e qualsiasi dato specifico del metodo di pagamento associato.

```
1  const methodData = [  
2    {  
3      supportedMethods: "basic-card",  
4      data: {  
5        supportedNetworks: ["visa", "mastercard"],  
6        supportedTypes: ["debit", "credit"],  
7      },  
8    },  
9    {  
10     supportedMethods: "https://example.com/bobpay",  
11     data: {  
12       merchantIdentifier: "XXXX",  
13       bobPaySpecificField: true,  
14     },  
15   },  
16 ];  
17
```

2.1.2 Details

I details contengono informazioni sulla transazione che l'utente è invitato a completare.

```

1  const details = {
2    id: "super-store-order-123-12312",
3    displayItems: [
4      {
5        label: "Sub-total",
6        amount: { currency: "USD", value: "55.00" },
7      },
8      {
9        label: "Sales Tax",
10       amount: { currency: "USD", value: "5.00" },
11       type: "tax"
12     },
13   ],
14   total: {
15     label: "Total due",
16     // The total is USD$65.00 here because we need to
17     // add shipping (below). The selected shipping
18     // costs USD$5.00.
19     amount: { currency: "USD", value: "65.00" },
20   },
21 };
22

```

2.1.3 Opzioni di spedizione

Qui vediamo un esempio di come aggiungere due opzioni di spedizione ai details.

```

1  const shippingOptions = [
2    {
3      id: "standard",
4      label: "Ground Shipping (2 days)",
5      amount: { currency: "USD", value: "5.00" },
6      selected: true,
7    },
8    {
9      id: "drone",
10     label: "Drone Express (2 hours)",
11     amount: { currency: "USD", value: "25.00" }
12   },
13 ];
14 Object.assign(details, { shippingOptions });
15

```

2.1.4 Modifiche condizionali alla richiesta di pagamento

Qui vediamo come aggiungere una tassa di elaborazione per l'utilizzo di una carta di credito. Si noti che richiede il ricalcolo del totale.


```

1  // Credit card incurs a $3.00 processing fee.
2  const creditCardFee = {
3    label: "Credit card processing fee",
4    amount: { currency: "USD", value: "3.00" },
5  };
6
7  // Modifiers apply when the user chooses to pay with
8  // a credit card.
9  const modifiers = [
10   {
11     additionalDisplayItems: [creditCardFee],
12     supportedMethods: "basic-card",
13     total:
14     {
15       label: "Total due",
16       amount: { currency: "USD", value: "68.00" },
17     },
18     data:
19     {
20       supportedTypes: "credit",
21     },
22   },
23 ];
24 Object.assign(details, { modifiers });
25

```

2.1.5 Options

Options contiene informazioni che lo sviluppatore ha bisogno dall'utente per eseguire il pagamento.

```

1  const options = {
2    requestPayerEmail: false,
3    requestPayerName: true,
4    requestPayerPhone: false,
5    requestShipping: true,
6  }
7

```

2.1.6 PaymentRequest

Dopo aver raccolto tutti i bit di informazioni prerequisite, ora possiamo costruirne uno PaymentRequest e richiedere che il browser lo presenti all'utente

```

1  async function doPaymentRequest()
2  {
3      try
4      {
5          const request = new PaymentRequest(methodData, details, options);
6          // See below for a detailed example of handling these events
7          request.onshippingaddresschange = ev => ev.updateWith(details);
8          request.onshippingoptionchange = ev => ev.updateWith(details);
9          const response = await request.show();
10         await validateResponse(response);
11     } catch (err)
12     {
13         // AbortError, SecurityError
14         console.error(err);
15     }
16 }
17 async function validateResponse(response)
18 {
19     try {
20         if (await checkAllValuesAreGood(response))
21         {
22             await response.complete("success");
23         }
24         else {
25             await response.complete("fail");
26         }
27     } catch (err)
28     {
29         // Something went wrong...
30         await response.complete("fail");
31     }
32 }
33 doPaymentRequest();
34

```

2.1.7 Gestione degli eventi e aggiornamento della richiesta di pagamento

Prima che l'utente accetti di effettuare il pagamento, al sito viene data l'opportunità di aggiornare la richiesta di pagamento in risposta all'input dell'utente. Ciò può includere, ad esempio, la fornitura di ulteriori opzioni di spedizione (o la modifica dei costi), la rimozione di articoli che non possono essere spediti a un determinato indirizzo, ecc.

```

1  const request = new PaymentRequest(methodData, details, options);
2  // Async update to details
3  request.onsshippingaddresschange = ev => {
4    ev.updateWith(checkShipping(request));
5  };
6  // Sync update to the total
7  request.onsshippingoptionchange = ev => {
8    const shippingOption = shippingOptions.find(
9      option => option.id === request.id
10   );
11   const newTotal = {
12     currency: "USD",
13     label: "Total due",
14     value: calculateNewTotal(shippingOption),
15   };
16   ev.updateWith({ ...details, total: newTotal });
17 };
18 async function checkShipping(request) {
19   try {
20     const json = request.shippingAddress.toJSON();
21     await ensureCanShipTo(json);
22     const { shippingOptions, total } = await calculateShipping(json);
23     return { ...details, shippingOptions, total };
24   } catch (err) {
25     return { ...details, error: 'Sorry! we can't ship to your address.' };
26   }
27 }
28

```

2.1.8 Segnalazione di errori a grana fine

Uno sviluppatore può utilizzare il `shippingAddressErrors`, membro del `PaymentDetailsUpdate`, per indicare che ci sono errori di convalida con attributi specifici di un `PaymentAddress`. Il `shippingAddressErrors` è un `AddressErrorFields`, i cui membri delimitano in modo specifico i campi di un indirizzo fisico errati e forniscono anche messaggi di errore utili per la visualizzazione all'utente finale.

```

1  request.onsshippingaddresschange = ev => {
2    ev.updateWith(validateAddress(request.shippingAddress));
3  };
4  function validateAddress(shippingAddress) {
5    const error = "Can't ship to this address.";
6    const shippingAddressErrors = {
7      cityError: "FarmVille is not a real place.",
8      postalCodeError: "Unknown postal code for your country.",
9    };
10   // Empty shippingOptions implies that we can't ship
11   // to this address.
12   const shippingOptions = [];
13   return { error, shippingAddressErrors, shippingOptions };
14 }
15

```

2.1.9 Postare la risposta di pagamento su un server

È previsto che i dati in un `PaymentResponse` verranno reindirizzati a un server per l'elaborazione. Per rendere questo più semplice possibile, `PaymentResponse` fornisce un `toJSON()` metodo che serializza l'oggetto direttamente in JSON. Ciò rende banale il POST del JSON risultante su un server che utilizza l'API Fetch.

```

1  async function doPaymentRequest() {
2      const payRequest = new PaymentRequest(methodData, details, options);
3      const payResponse = await payRequest.show();
4      let result = "";
5      try {
6          const httpResponse = await fetch("/process-payment", {
7              method: "POST",
8              headers: { "Content-Type": "application/json" },
9              body: payResponse.toJSON(),
10         });
11         result = httpResponse.ok ? "success" : "fail";
12     } catch (err) {
13         console.error(err);
14         result = "fail";
15     }
16     await payResponse.complete(result);
17 }
18 doPaymentRequest();
19

```

3 Codice d'esempio per pagina demo

Implementazione PaymentRequest API su una pagina d'esempio.

3.0.1 Costruttore

Iniziamo con il costruttore. L'oggetto PaymentRequest è costruito passando i seguenti parametri:

- **methodData**: una serie di dizionari che contiene gli identificativi del metodo di pagamento e tutti i dati pertinenti; un identificativo del metodo di pagamento è una stringa che identifica un metodo di pagamento supportato
- **dettagli**: informazioni sulla transazione, come gli elementi pubblicitari in un ordine
- **opzioni**: informazioni aggiuntive che il Wallet potrebbe dover raccogliere

Nel seguente snippet, stiamo consentendo agli utenti di pagare con qualsiasi carta di debito o di credito appartenente alle reti Visa, MasterCard o Amex. L'oggetto dettagli contiene l'importo totale parziale, l'imposta sulle vendite e il totale dovuto. Questi dettagli verranno mostrati all'utente nel portafoglio. Bisogna tenere presente che l'API non aggiunge elementi o calcola l'imposta sulle vendite: spetta al commerciante fornire le informazioni corrette. In questo esempio, stiamo vendendo un bene fisico, quindi chiediamo al portafoglio di ritirare l'indirizzo di spedizione del cliente.

```

1  var methodData = [
2    {
3      supportedMethods: ['basic-card'],
4      data: {
5        supportedNetworks: ['visa', 'mastercard', 'amex'],
6        supportedTypes: ['credit']
7      }
8    }
9  ];
10
11  var details = {
12    displayItems: [
13      {
14        label: "Sub-total",
15        amount: { currency: "USD", value : "100.00" } // US$100.00
16      },
17      {
18        label: "Sales Tax",
19        amount: { currency: "USD", value : "9.00" } // US$9.00
20      }
21    ],
22    total: {
23      label: "Total due",
24      amount: { currency: "USD", value : "109.00" } // US$109.00
25    }
26  };
27
28  var options = {
29    requestShipping: true
30  };
31
32  var payment = new PaymentRequest(methodData, details, options);
33

```

3.0.2 Visualizzazione dell'interfaccia utente, elaborazione del pagamento e visualizzazione dei risultati

Una volta creato l'oggetto `PaymentRequest`, è possibile attivare il browser per visualizzare il wallet con `request.show()`.


I clienti possono quindi selezionare le informazioni di pagamento, l'indirizzo di spedizione e altri campi appropriati, quindi fare clic su Paga quando è pronto. A questo punto, gli utenti dovranno verificare la loro identità. In caso di esito positivo, ciò soddisferà la promessa `request.show()` e restituirà al sito Web tutte le informazioni che il cliente ha fornito al Portafoglio. Per il metodo di pagamento con carta di base, l'oggetto risultato conterrà il nome del titolare della carta, il numero della carta, il mese di scadenza e altri campi pertinenti. Il commerciante può quindi utilizzare queste informazioni per elaborare la transazione sul back-end. Dopo che la risposta è tornata dal server, è possibile utilizzare `result.complete('success')` per visualizzare la schermata di successo nel Portafoglio e `result.complete('fail')` per indicare una transazione fallita.

Microsoft Wallet


✕

Confirm and pay

www.webstoreurl.com



Pay with

 John Smith •• 5567

▼

Ship to

John Smith
13311 NE 100th St, #100
Seattle, WA 98100

▼

Shipping options

Standard - FREE 5-6 Business days

▼

Email receipt to

johnsmith@outlook.com

▼

Total (USD) [Show details](#) ▼

\$345.00

Pay

Figura 2: Wallet dopo la chiamata request.show()

```

1  // Mostra l'interfaccia utente nativa
2  payment.show()
3  // Quando la promessa è soddisfatta, passa i risultati al tuo server per l'elaborazione
4  .then(result => {
5      return process(result).then(response => {
6          if (response.status === 200) {
7              // Mostra che la transazione ha avuto successo nell'interfaccia utente
8              return result.complete('success');
9          } else {
10             // Mostra nell'interfaccia utente nativa che la transazione ha avuto esito
                negativo
11             return result.complete('fail');
12         }
13     }).catch((err) => {
14         console.error('User rejected request', err.message)
15     });
16 });
17

```

3.0.3 Ascoltando gli eventi

Il prezzo potrebbe cambiare in base all'indirizzo di spedizione e alle opzioni di spedizione selezionate dal cliente. È possibile ascoltare tali modifiche con gli eventi `shippingaddresschange` e `shippingoptionchange` per ricalcolare di conseguenza i prezzi.

```

1  payment.addEventListener("shippingaddresschange", function (changeEvent) {
2      // Elabora la modifica dell'indirizzo di spedizione
3  });
4
5  payment.addEventListener("shippingoptionchange", function (changeEvent) {
6      // Modifica delle opzioni di spedizione del processo (ad esempio "spedizione in
        giornata")
7  });
8

```

3.0.4 Rilevamento di funzionalità

I siti possono essere rilevati per l'API di richiesta di pagamento, inoltrare l'utente a un'esperienza legacy, basata su moduli, se non è disponibile.

```

1  // Inoltra utente alla verifica basata su modulo se PaymentRequest non è
    disponibile
2  if (!window.PaymentRequest) {
3      window.location.href = '/form-based-checkout';
4      return;
5  }
6

```

Ecco un esempio di implementazione minima di questo codice:

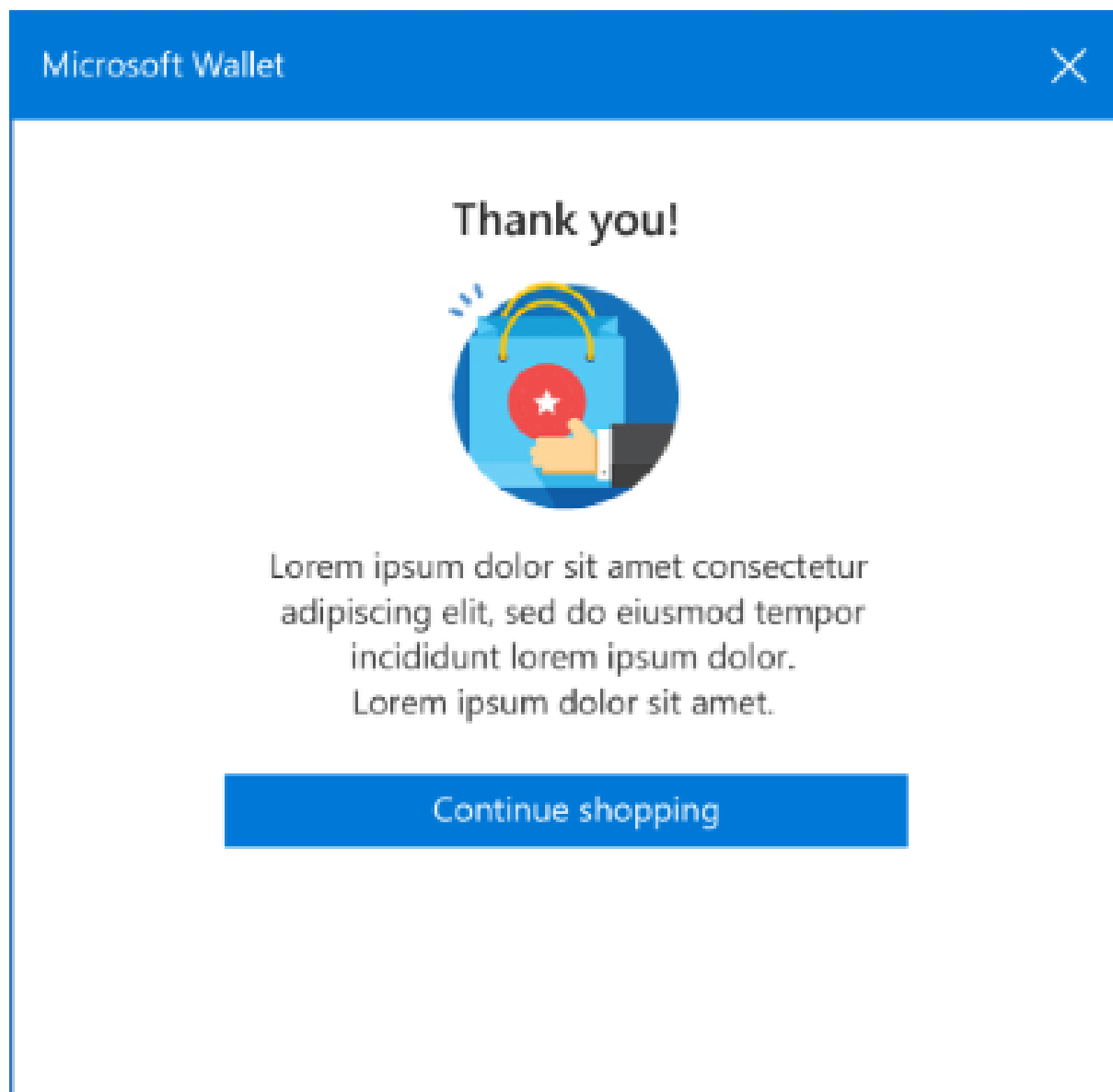


Figura 3: Wallet in caso di successo

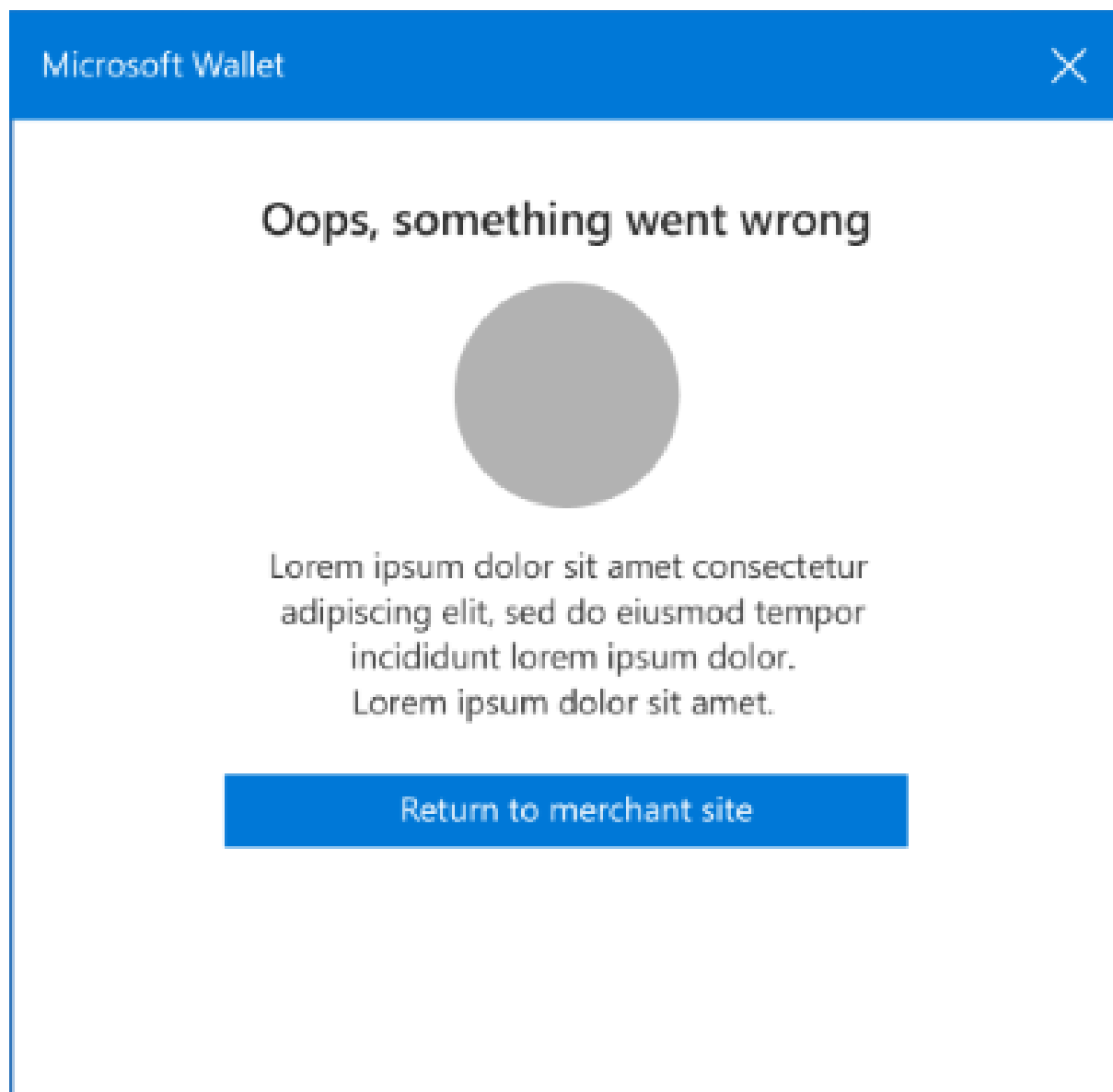


Figura 4: Wallet in caso di fail

```

1  function startPaymentRequestCheckout() {
2      var methodData = [
3          {
4              supportedMethods: ['basic-card'],
5              data: {
6                  supportedNetworks: ['visa', 'mastercard', 'amex'],
7                  supportedTypes: ['credit']
8              }
9          }
10     ];
11
12     var details = {
13         displayItems: [
14             {
15                 label: "Sub-total",
16                 amount: { currency: "USD", value : "100.00" }, // US$100.00
17             },
18             {
19                 label: "Sales Tax",
20                 amount: { currency: "USD", value : "9.00" }, // US$9.00
21             }
22         ],
23         total: {
24             label: "Total due",
25             amount: { currency: "USD", value : "109.00" }, // US$109.00
26         }
27     };
28
29     var options = {
30         requestShipping: true
31     };
32
33     var request = new PaymentRequest(methodData, details, options);
34
35     //Show the Native UI
36     request.show()
37     //When the promise is fulfilled, pass the results to your server for processing
38     .then(result => {
39         return process(result).then(response => {
40             if (response.status === 200) {
41                 //Show that the transaction was successful in the UI
42                 return result.complete('success');
43             } else {
44                 //Show in the Native UI that the transaction failed
45                 return result.complete('fail');
46             }
47         })
48     });
49
50     request.addEventListener("shippingaddresschange", function (changeEvent) {
51         // Process shipping address change
52     });
53
54     request.addEventListener("onshippingoptionchange", function (changeEvent) {
55         // Process shipping option change (e.g. "one day shipping")
56     });
57 }
58 document.getElementById('#checkout').addEventListener('click',
59     startPaymentRequestCheckout)

```

4 Compatibilità web

Desktop		Mobile				
Feature	Chrome	Edge	Firefox (Gecko)	Internet Explorer	Opera	Safari (WebKit)
Basic support	61	(Yes)	No support ^[1]	?	No support	?

Figura 5: Compatibilità desktop

Desktop		Mobile					
Feature	Android Webview	Chrome for Android	Edge	Firefox Mobile (Gecko)	IE Mobile	Opera Mobile	Safari Mobile
Basic support	No support	51	(Yes)	No support ^[1]	?	No support	?

Figura 6: Compatibilità mobile

5 Conclusioni

L'API di richiesta di pagamento fornisce un potente strumento per i commercianti per migliorare la conversione del checkout sul Web e per offrire ai clienti un'esperienza di acquisto più piacevole e conveniente. Questa API è un ottimo esempio della potenza e della flessibilità della piattaforma web, ed è sulla strada per un'ampia interoperabilità, con Chrome per Android che supporta l'API a partire da Chrome 54.