Payment Request UX considerations

By Mustafa Kurtuldu

Design Advocate at Google • Material Design, UX & Design Sprints

The Payment Request API was fundamentally built for users, helping them get through burdensome checkout flows on mobile efficiently, while providing merchants with a higher conversion rate.

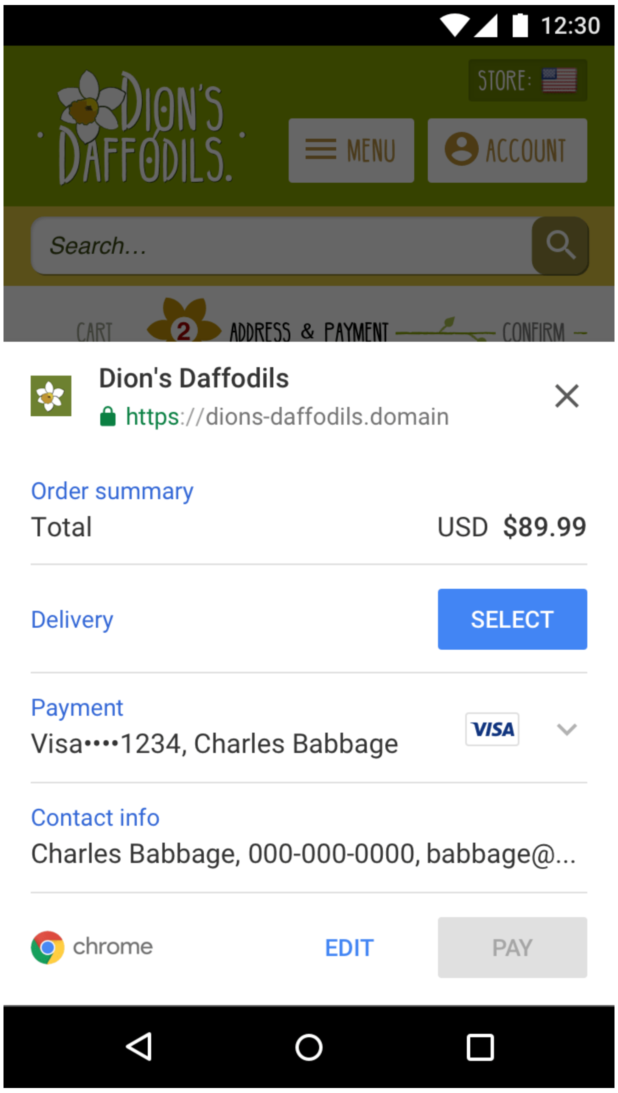
In a study conducted by Google, we found that **65.9% of users stopped partway through purchase on mobile devices**. Of those asked in the study as to why they abandoned the purchase, the most common responses ranged from the form being difficult to read, difficult to understand or was way too long.

To solve these challenges the Payment Request API creates a simpler flow. The API will pre-populate payment details for the user allowing users to buy items with a simple tap or click.

One way to think about the Payment Request API is that it is essentially a way for a browser to manage a user's payment methods, making it easier to make payments on the web.

The two view states

In the Chrome UI, the Payment Request API has two states: the *'receipt'* view and the edit view. The *'receipt'* view (shown at left, below) partially obscuring the merchant's site. This allows the user to take a quick glance at their payment details. Then there is the edit view (shown at right, below), which expands full screen and allows the user to edit each section. Once the user taps or clicks a call to action from the *'receipt'* view they will enter the edit mode and will be forced to continue there.



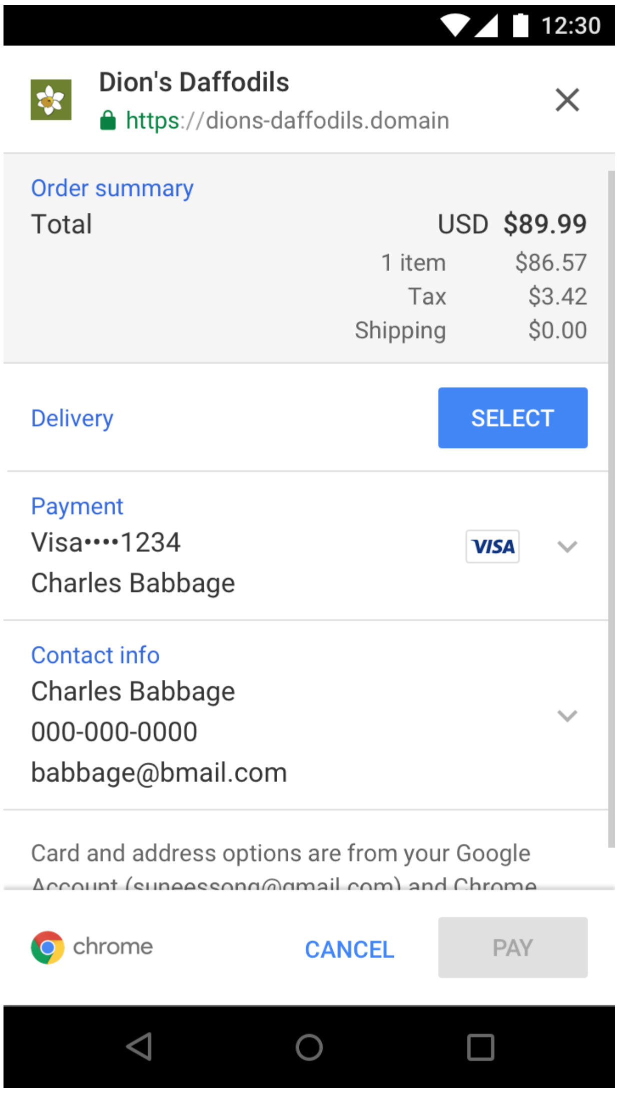
Dion's Daffodils
https://dions-daffodils.domain

Order summary
Total USD $89.99

Delivery SELECT

Payment
Visa••••1234, Charles Babbage

Contact info
Charles Babbage, 000-000-0000, babbage@...


Dion's Daffodils
https://dions-daffodils.domain

Order summary
Total USD $89.99

1 item	$86.57
Tax	$3.42
Shipping	$0.00

Delivery SELECT

Payment
Visa••••1234
Charles Babbage

Contact info
Charles Babbage
000-000-0000
babbage@bmail.com

Card and address options are from your Google Account (suneessong@gmail.com) and Chrome

Basic checkout flow

In this example a user navigates to a site, selects an item and goes through the payment request checkout flow. Once a user clicks buy, the browser takes over. If the user has previously entered details such as their preferred payment method, these will appear as the defaults. Otherwise the user can add these. Depending on your particular use case you can have defaults auto selected or force the user to fill them in.

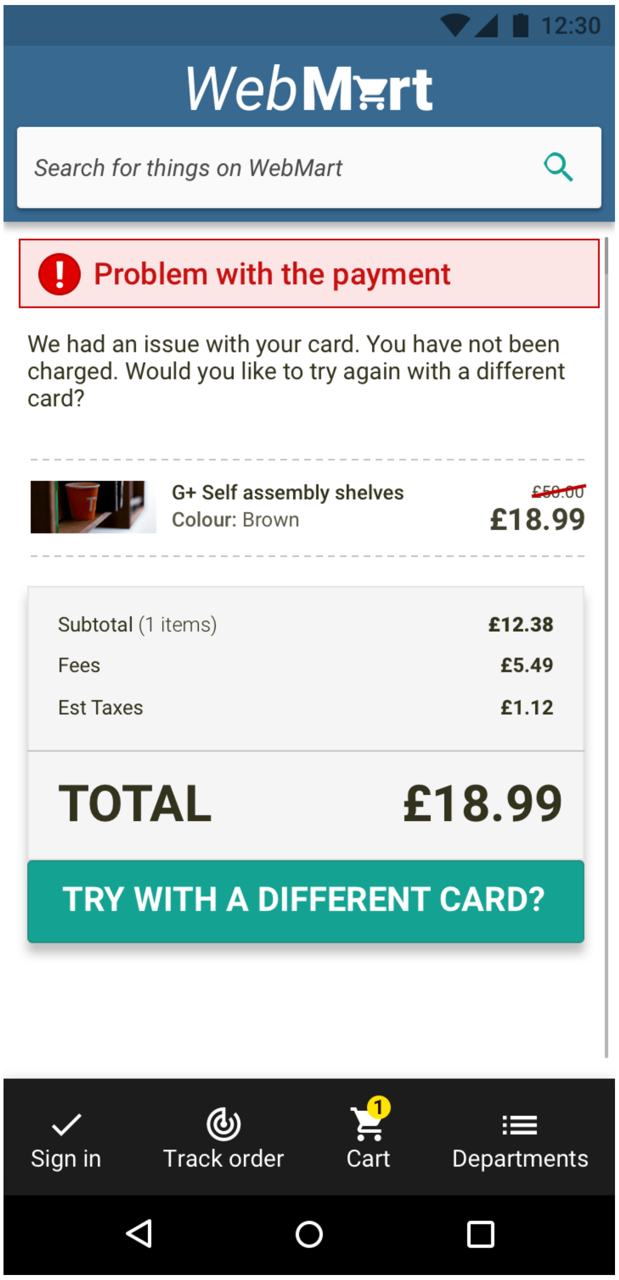
Checkout guidance

Don't block checkout by requiring signup. One of the core benefits of having users already signed in is that you already have payment methods stored for them. Since Payment Request provides easy access to the users payment details, including email and phone number, you can simply request these details and avoid requiring users to sign in.

Gracefully handle the UX flow

Chrome handles some aspects of the processing, cancellation, and success but as a best practice, you should signal to the user what the status of their checkout is. For example:

- By Informing the user if their payment has been aborted.
- Prepare the user for the checkout process by providing UI that will tell them that you will be asking for payments.
- Keep the user informed of progress and state of the checkout is by using steppers.
- Signaling to the user that payment was complete and successful.


12:30

WebMart

Search for things on WebMart

Problem with the payment

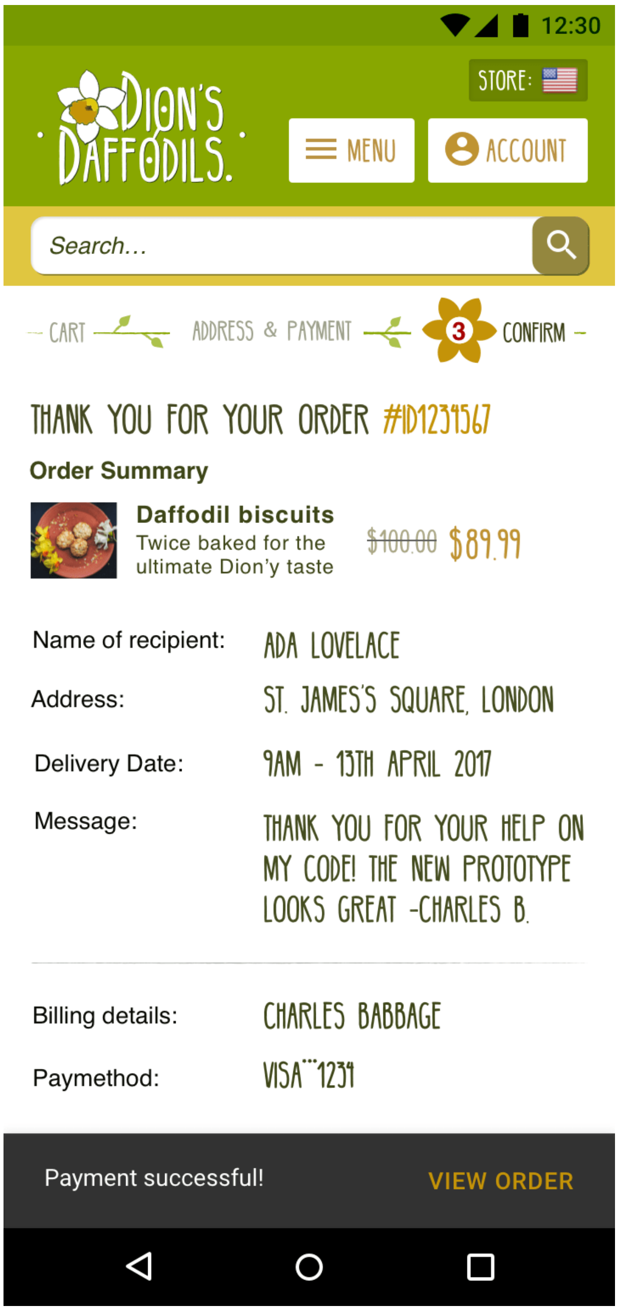
We had an issue with your card. You have not been charged. Would you like to try again with a different card?

G+ Self assembly shelves
Colour: Brown
~~£50.00~~
£18.99

Subtotal (1 items)	£12.38
Fees	£5.49
Est Taxes	£1.12
TOTAL	£18.99

TRY WITH A DIFFERENT CARD?

Make sure error messages are clear.



THANK YOU FOR YOUR ORDER #ID1234567

Order Summary

Daffodil biscuits
Twice baked for the ultimate Dion'y taste ~~$100.00~~ $89.99

Name of recipient:	ADA LOVELACE
Address:	ST. JAMES'S SQUARE, LONDON
Delivery Date:	9AM - 13TH APRIL 2017
Message:	THANK YOU FOR YOUR HELP ON MY CODE! THE NEW PROTOTYPE LOOKS GREAT -CHARLES B.
Billing details:	CHARLES BABBAGE
Paymethod:	VISA***1234

Payment successful! VIEW ORDER

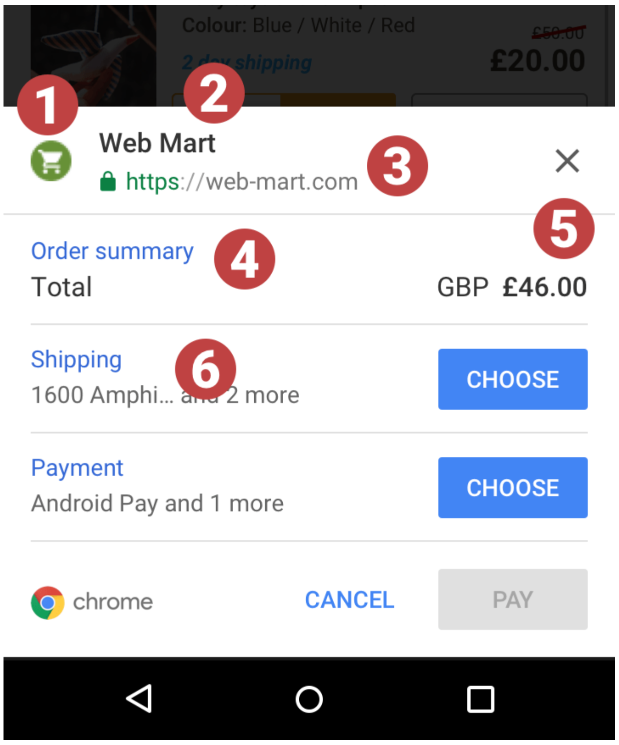
This stepper shows the progress from Cart to Address & Payment to Confirm.

What is customisable

Some of the UI is customisable with some exceptions such as the colour of button and their respective text labels.

Note: This example is for the Chrome browser. Other browsers may contain different UI elements as well, or even labels for buttons such as 'Pay' and 'Cancel'.

Another consideration is that devices come in many sizes, so space maybe at a premium, so keep the labels short and focused on brand recognition.

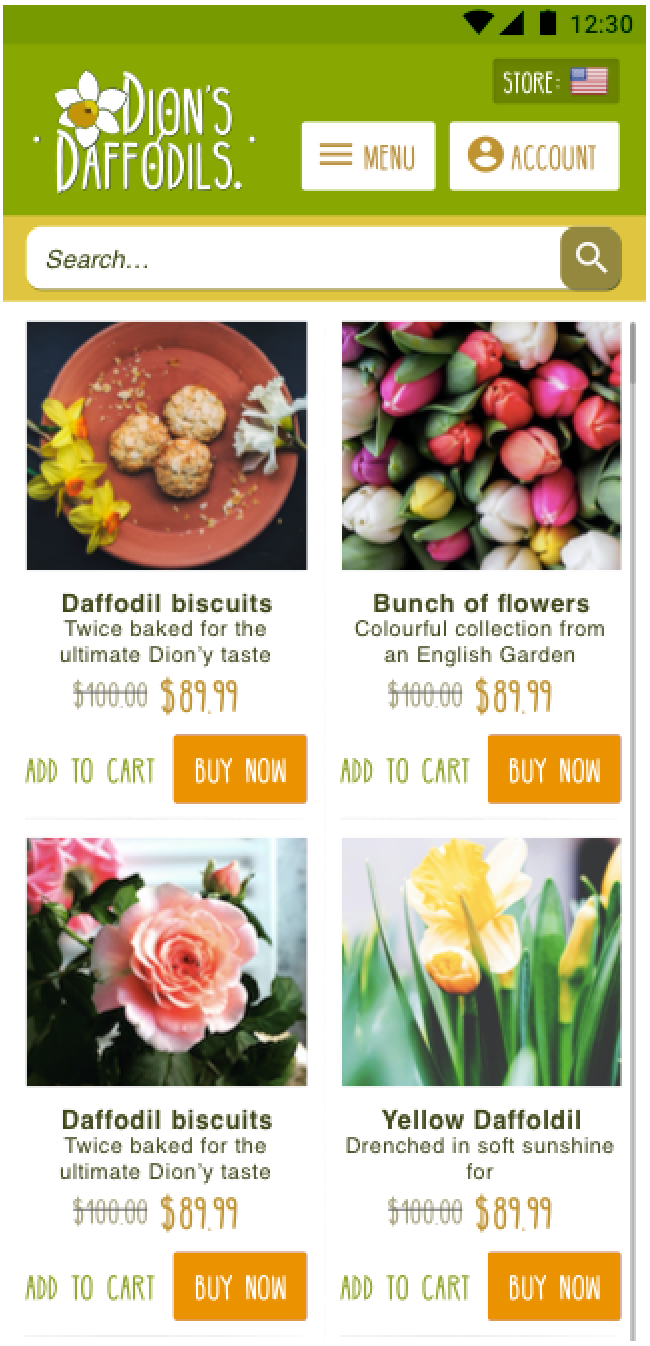


1. The icon is taken from the favicon. Favicon is hi-res and recognizable for the store. For best practice, provide multiple icons at different sizes to allow the browser to pick the most appropriate size. You can learn more here.
2. This is taken from the `<title>` tag. This is your strongest chance to provide a human-readable indication of what the user is buying and who they're buying from.
3. This is the domain/URL.
4. This text label can be modified via the Payments Request API.
5. The price and currency is taken from the total.
6. You can select a heading from a preset list: Shipping, Pickup OR Delivery.

General UX advice

Eliminate user steps

Try to limit the amount of work it takes for a user to make a purchase. A simple UI with clear text labels will give the users a perceived increase in speed. For example using a *"Buy Now"* button allows the user to make a simple one tap purchase without forcing them to create an account or populate a shopping cart. The faster a user can checkout, the more satisfied they become. This means the chance of them completing the process greatly increases and as does the chance of repeat business. Allowing users to purchase an item with a *"Buy now"* or *"Express Checkout"* button (versus *"Adding to cart"*), will improve their perceived speed, thereby making your checkout a success.



Add *'Buy now'* buttons where possible to create quick checkout experience

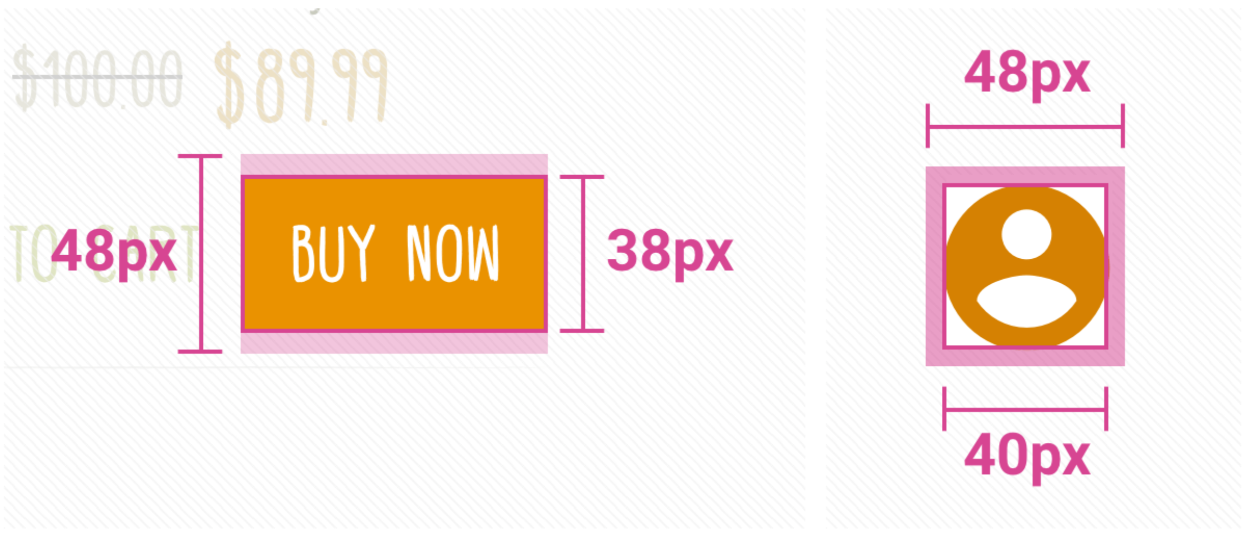
UI affordance

Users in the real world often wrap up their mobile devices with sleeves or bumpers. Think of ergonomics and considerations - bumpers can prevent users from hitting the touch targets if they are too close to the screen giving them the impression that they are tapping buttons when they are not therefore presenting the illusion that the site and app is slower than it actually is.

Touch targets

Make sure the touch targets extend beyond the visual bounds of an element. For example, a 24 x 24 pixel icon should have a 48 x 48 pixel touch target. .

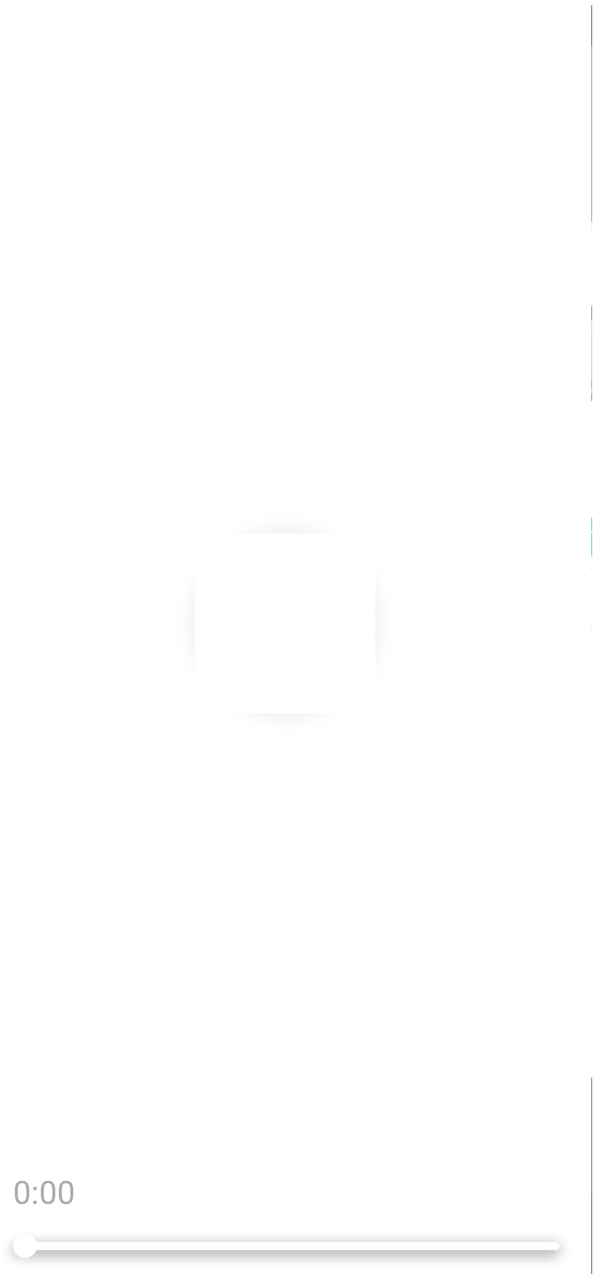
A 48 x 48 px touch target is roughly 9mm and the recommended target is between 7-10mm. You may start using the image's own dimensions as the target target if the image is larger than 48 x 48 px. It may be appropriate to use even larger touch targets to accommodate users with special needs such as those with motor disability skills.



By giving a minimum of 48px touch target you can help users tap or click smaller UI elements.

User feedback mechanisms

As well as providing confirmation to users that a purchase was successful, provide feedback to reassure them of the status of their experience. One example of feedback is a "Snackbar" UI, that appears once an action is completed. The average reader can read and comprehend between 200-400 words per minute. Depending on the length of the string make sure you leave the UI on screen for long enough for the user to read, roughly 3 seconds for five word sentence.

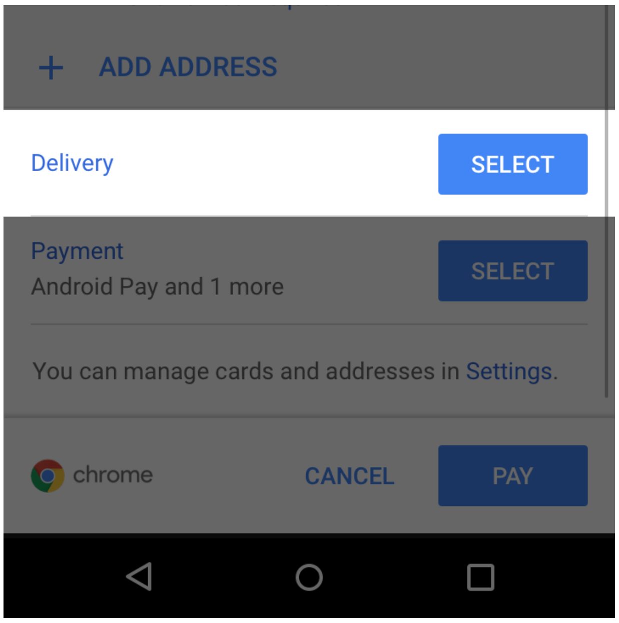


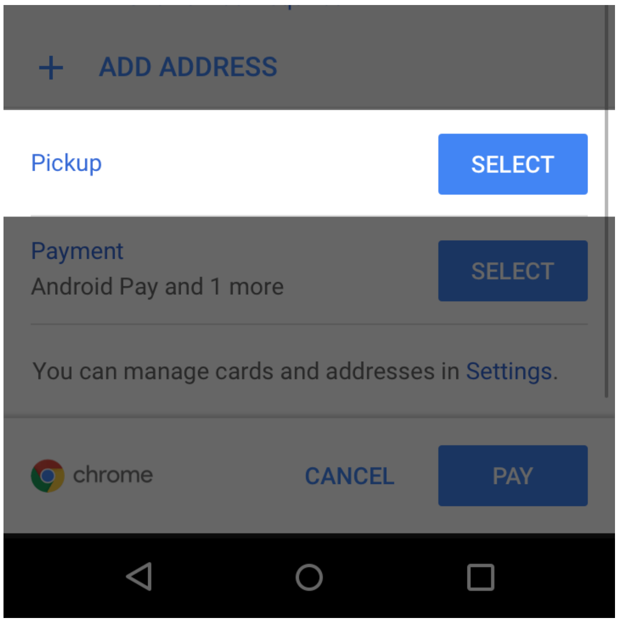
Performance

In a different study conducted at Google, we found 70% of users dropped from a checkout due to the first dialogs not rendering fast enough. Make sure you test for poor network conditions in order to improve your checkout flow by using lighthouse and DevTools network emulation.

Language

Improved language in the call to action buttons help users understand what actions were being taken in the flow. It is important that they are consistent in the checkout flow. Make sure you use the labels that make the most sense.





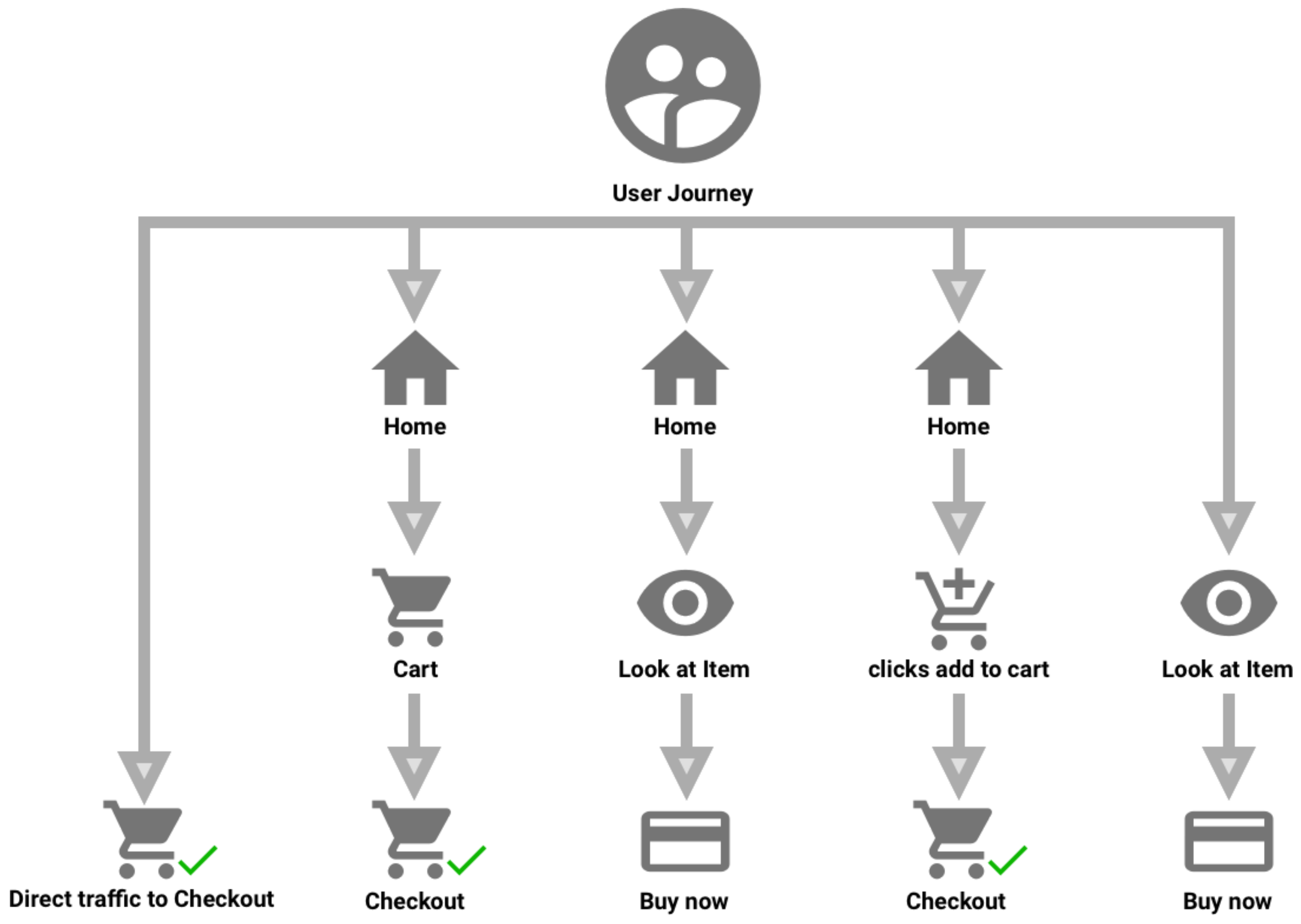
Depending on the context the label *"Pickup"* may be more appropriate than the label *"Delivery"*.

Provide fall back options to the user

Don't block a purchase on Payment Request. If isn't supported by the browser, or fails, seamlessly fallback to the default checkout rather than an error message when the Payment Request API is not available.

It is a good idea to create a user flow of all of the possible ways a user can get to the checkout forms. Common entry points that you need to design for;

- The user is immediately directed to checkout.
- The user starts at the home page, views their cart, and is directed to checkout.
- The user starts at the home page, looks at an item, and is directed to buy now.
- The user starts at the home page, clicks to add an item to their cart, and is directed to checkout.
- The user starts at the item and is directed to the checkout.



This is an illustrative example of the above, different paths a user takes when checking out

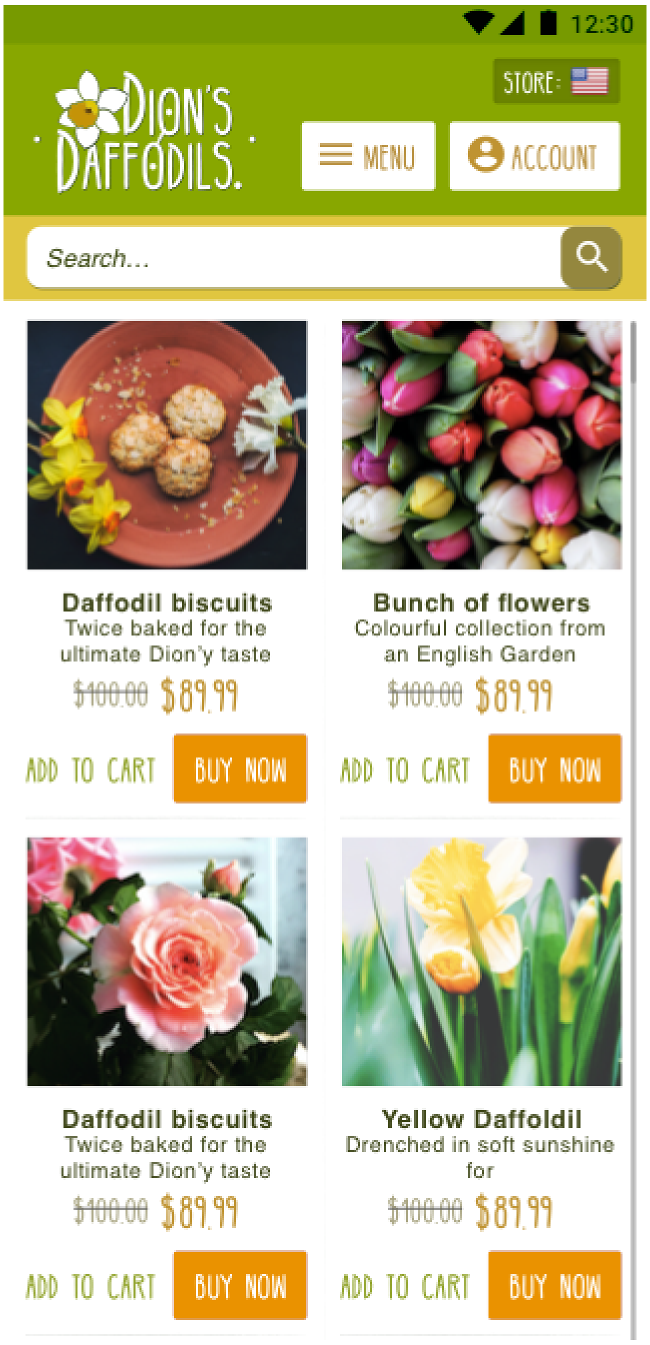
Example flows

Guest checkout

In this example we allow the user to checkout, then email them an ID code and receipt. Alternatively you can offer a button that calls `window.print();` and gives the user a PDF.

Buy now UI

In this example every item has a "Buy now". When a user taps or clicks "Buy now" they begin the checkout process, first by asking for specific delivery dates and a custom message. The next step in the process launches the Payment Request API which is followed by a confirmation page.



Add *"Buy now"* buttons where possible to create quick checkout experience

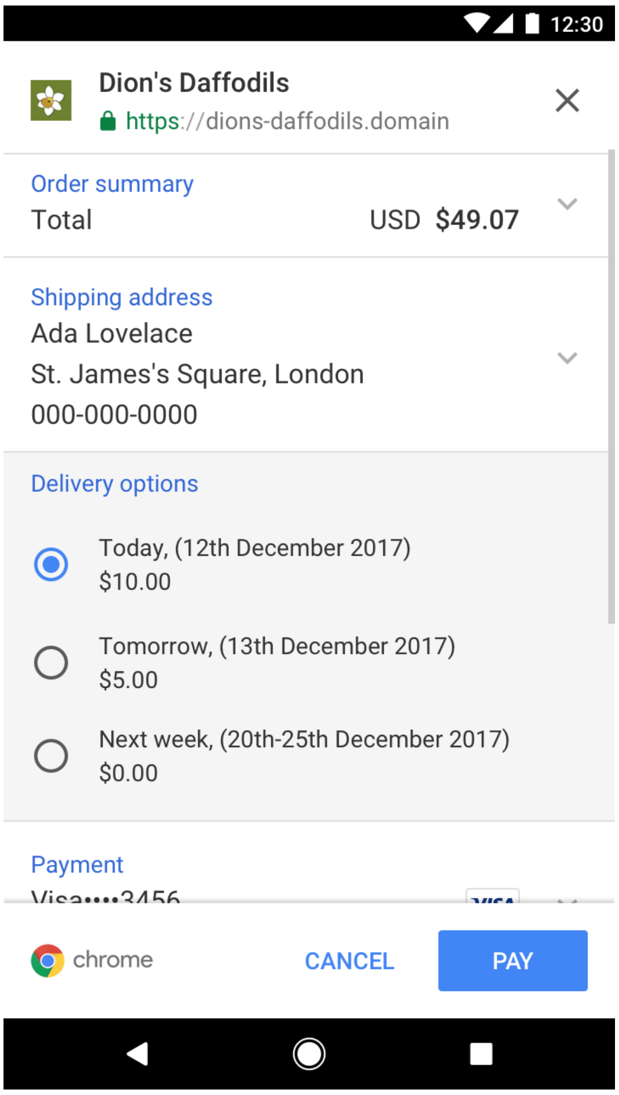
Delivery

Although it is technically possible to auto-select the user's default address it is best practice to allow the user to select their preferred address themselves. This will allow the merchant verify that the address is correct and deliverable. By allowing the user to change the address, you can update and alter the shipping options. For example if the user is local you can offer local shipping rates, but if the user selects an international address, you can alter the shipping options for international rates.

By auto selecting the address you can create needless friction. For example, if the default address is somewhere you cannot deliver or ship to, it will only be picked up once the user has triggered the actual pay action. Avoid creating a jarring experience at all costs as this can cause a user to abandon the purchase altogether.

Shipping options

Give the user delivery options according to actual dates. Contextual data such as the date will give the user context and further reassurance of when their items will be delivered.



12:30

Dion's Daffodils
https://dions-daffodils.domain

Order summary
Total USD $49.07

Shipping address
Ada Lovelace
St. James's Square, London
000-000-0000

Delivery options

[✓ Today, (12th December 2017) $10.00]

Tomorrow, (13th December 2017) $5.00

Next week, (20th-25th December 2017) $0.00

Payment
Visa••••3456

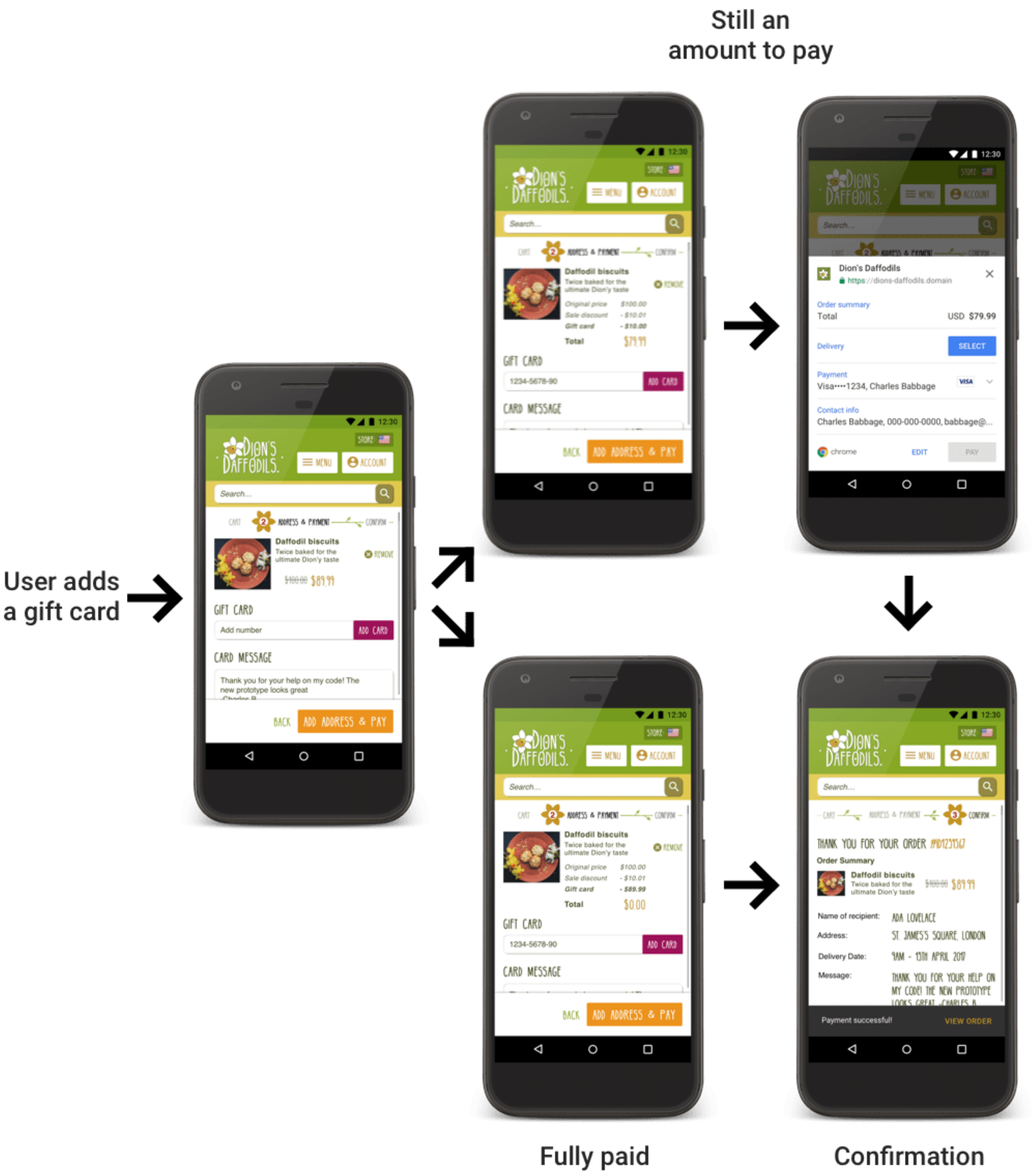
Give dates to help a user plan and customize their experience.

Adding additional information

There are many use cases where the merchant may require the user to add additional information before or after the transaction. Such as loyalty cards numbers or discount coupons.

Gift cards

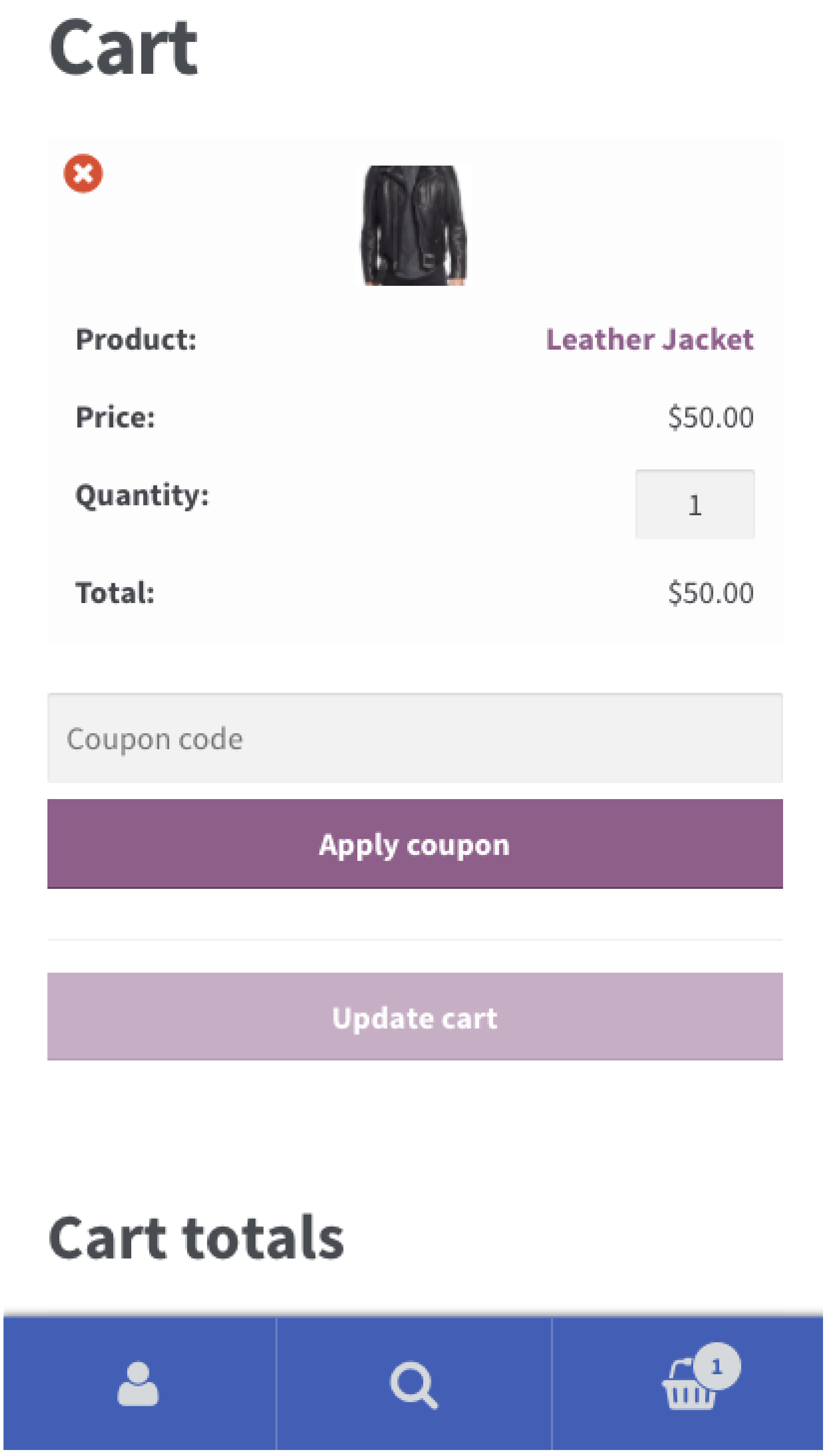
Gift cards do not work directly with the Payment Request API so we recommend adding a field that allows the user to enter their card code beforehand. Once they add the gift card if there is still more to pay you can launch the Payment Request API. The illustration below demonstrates this flow.



Customizing the Payment Request API flow

Before calling Payment request API

The Payment Request API doesn’t deal with vouchers and loyalty codes directly. Instead, give the user the ability to add any codes or loyalty card schemes before you launch the Payment Request API. In practice, this will mean that when a user first clicks *"Buy now"*, they will be taken to a checkout page where they can add a coupon or loyalty card number first. Below is an example of this.



Cart

Product:	Leather Jacket
Price:	$50.00
Quantity:	1
Total:	$50.00

Cart totals

In this example WooCommerce the user adds any coupon code before they complete the transaction. (The demo website works only with a fake credit card number "4242 4242 4242 4242.")

If a merchant's legal team requires a legal text box such as a Terms of Service (TOS) to be displayed during the checkout process we recommend that this appears before the Payment Request API flow begins. Ideally you would want to minimize the steps and actions taken by the user by keeping the TOS next to the buy button or a link. However there are some cases where this is not possible, so one solution is after the user clicks the "Buy Now" button, provide an additional step in the processes asking for acceptance of the TOS. The next step is to launch the PR Payment Request flow.

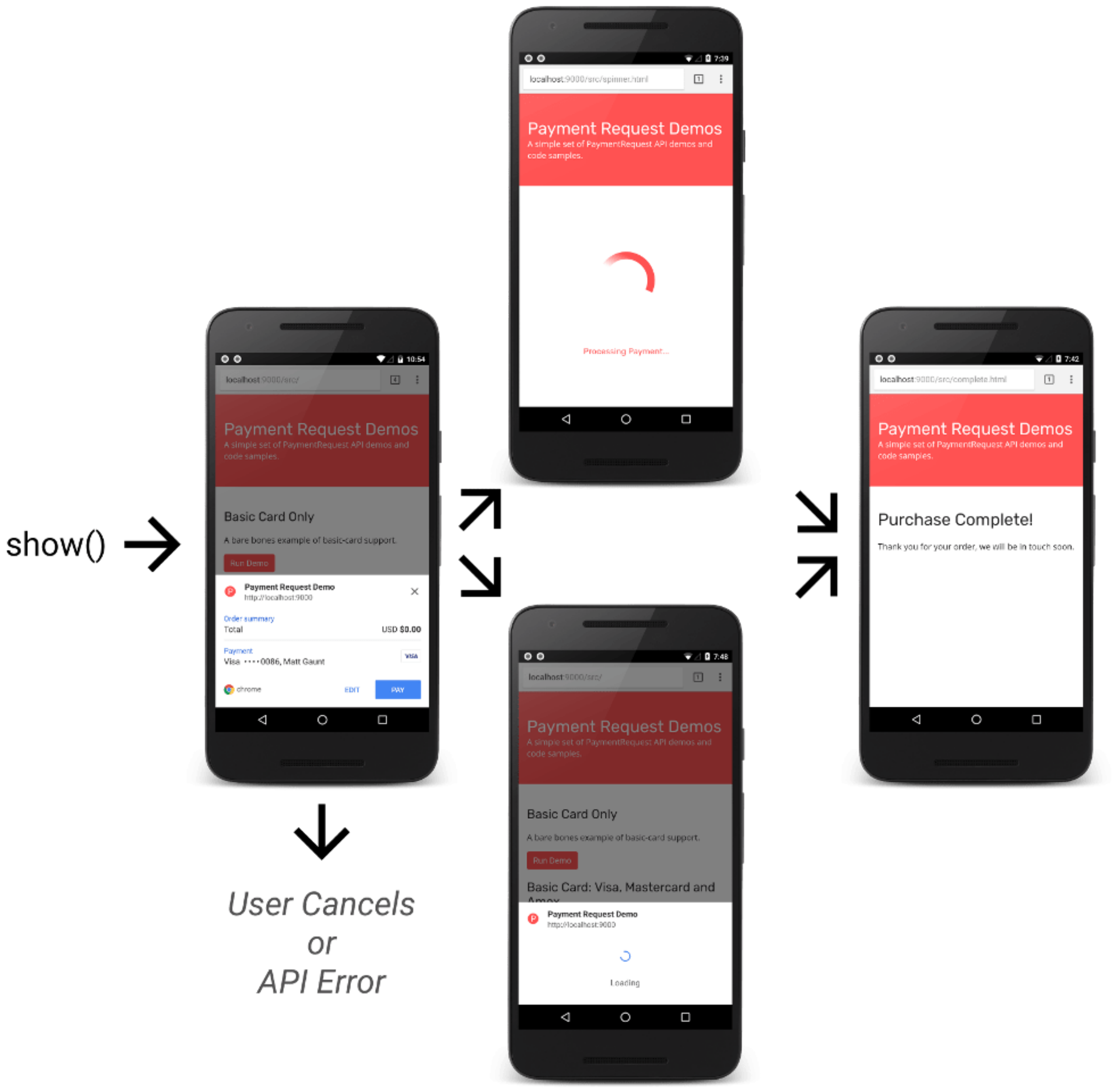
Midway through the Payment Request

You can offer extra UI elements after the payment options have been selected. So instead of showing the API's default spinner, you can show your own, such as in the next example. Then once you have processed the payment details, you can offer up any UI you choose.

So in this visual example: * The user taps "Buy now" and the merchant calls the function show();

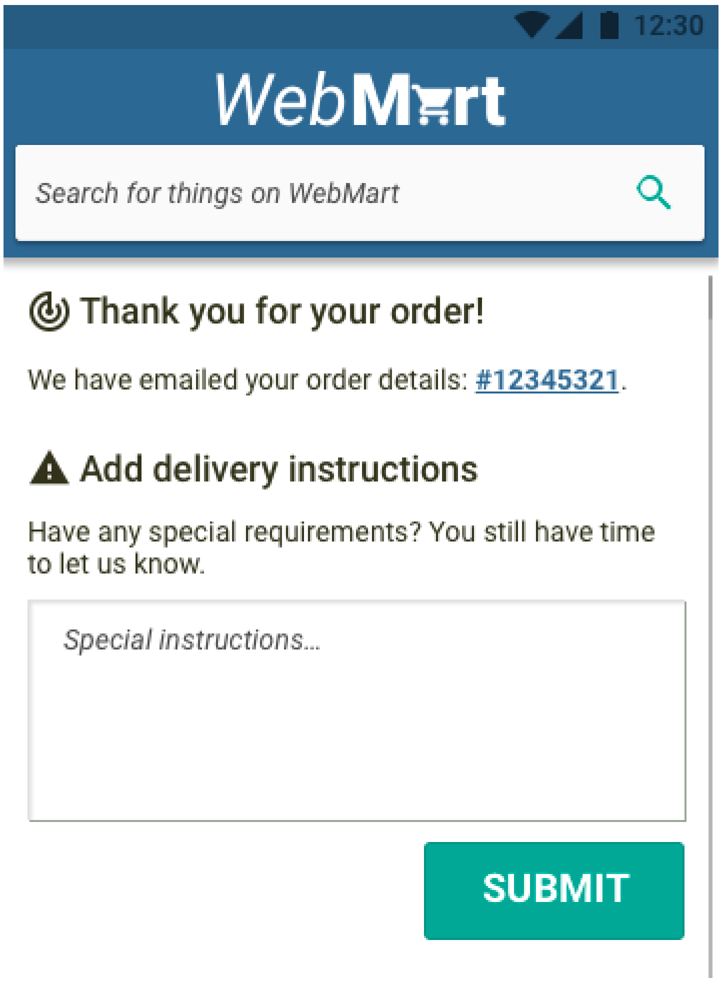
- Then the merchant either processes the payment with the Payment Request API OR their own UI.
- If all conditions are met the merchant serves up a confirmation page.

You can learn more about this technique here



After payment / purchase has been processed.

There may be occasions where the merchant give the user the opportunity to add non-standard information from the user such as delivery instructions or perhaps some feedback on the shopping experience. This can not be done directly within the Payment Request API, however it is possible to show a UI element such as a `textfield` if the payment has been successful.



Thank you for your order!

We have emailed your order details: #12345321.

Add delivery instructions

Have any special requirements? You still have time to let us know.

Code example:

```
request.show()
.then(function(paymentResponse) {
    // Process payment here.
    // Close the UI:
    paymentResponse.complete('success').then(function() {
        // Request additional shipping address details.
        const additionalDetailsContainer = document.getElementById('additional-de
        additionalDetailsContainer.style.display = 'block';
        additionalDetailsContainer.focus();
    }).catch(function(error) {
        // Handle error.
    });
})
.catch(function(error) {
    // Handle error.
});
```

You can find code for more unique use cases here.

Existing users

For existing customers you can still use the Payment Request API, but you may need to adjust the experience and present new UI to your existing signed-in customers. For example you may introduce the Payment Request API with an "Express pay" button, a "Pay with new card" button or a "Ship to new address" button.

The goal is to show that the user can quickly buy things on the merchant's site without any fuss. The Payment Request API is quick and efficient but existing users will need some education with this new payment flow so focus on introducing the new UX patterns with text labels and UI that express speed and efficiency.

Code demos and browser support

You can experiment with the various features and UX patterns with this demo. Payment Request API is currently supported by;

- Chrome 53 and above on Android
- Chrome 61 and above on Desktop & iOS
- Edge 15 and above on Desktop

You can track the latest status for all other browsers here on caniuse.com.

Stickersheet

To create your own flows, you can download the Payment Request API sticker sheet from our Github Repo.