**Implementation strategy**

**US216: Customer Native App - card.io implementation**

This story is to add card.io integration into the customer native app.

card.io is a native library for using a mobile device's camera to read the credit card number from a credit card. It is currently being used in the admin/driver native app.

**Changes to Customer Native App**

There are a few places where credit card information can be entered. The following places need to be updated:

* Register Payment Method - Accessed from the side menu ("Payment" menu item) - Then tap the "ADD PAYMENT METHOD" button (or the + button in the upper-right corner).
* Payment screen - This is accessible after the user has requested their vehicle. When the user submits payment, they will be asked if they want to use a registered card, or enter a different card. If they choose a different card, it shows the card info screen.
  + One Time Payment - The user might choose to not register the new card information. In this case just submit the payment.
  + Register Payment - The user might choose to register the new card information. In this case, register the card and then submit the payment.

Existing implementation:

* Open a WebView component that renders a web page (hosted by the Payment Service).
* When the user enters the credit card information and submits the web page, it will silently make a call to the Intuit token API to get a payment token.
* If the token was created successfully, the web page will publish a message that can be read by the parent frame (in this case our WebView component).
* The WebView listens for the message that contains the payment token. When the token is created, it will call the Vark Payment Service API to either register the card or pay with a one-time transaction.

New Implementation:

* Replace the WebView component with a new component that uses the card.io library. Launch the card.io scanner (launches the camera)
* The card.io library has a callback that you can listen to when the card data is read. Once it is done (either successfully reading the card, or canceled for manual entry), show a screen with the following fields:
  + Customer Name - Default to the customer's name from their user information (see the profile screen to see the API calls to get this information)
  + Card Number - Default should be the card.io value. If the camera couldn't read it, or if the user cancels card.io scanning, leave this blank.
  + CVV -
  + Expiration Date - Dropdowns for month and year. Default should be the card.io value. If the camera couldn't read it, or if the user cancels card.io scanning, leave this as the current month and year.
  + (Note: card.io will only read the card number and expiration date, so the other fields must be manually entered)
* Submit the card information to the Intuit Token API to get a payment token.
* Once you have the payment token, you can use the Vark Payment Service API as it is used in the existing implementation. (Register payment or pay with one-time payment).

**Notes/Tips**

Take a look at the admin native app. There is a card.io implementation that shows how to use it in the way that I described above. It can be found at  **admin-native/comps/customer-cc-payment-comp.js**

You can add card.io to the project by using the npm install command and then linking it to the project.

> npm install -s react-native-awesome-card-io

> react-native link react-native-awesome-card-io

Check package.json to make sure it was added to the dependencies. The admin app has been tested with version ^0.7.0, so if you run into any issues with react-native compatibility, you may want to check this version first.

**Make sure that any credit card information is only submitted to the Intuit Token API. Do not send any credit card information to Vark's backend services. This is to ensure that we remain PCI compliant. (It is okay to send the token that we get from the Intuit Token API, but make sure to never submit the credit card number).**