Week 8 Assignment

Payment Processing System Software Implementation

for

Master of Science

Information and Communications Technology

Distributed Computing

ICT 4310-1

Richard Magill

University of Denver University College

May 8, 2022

Faculty: Mike Prasad

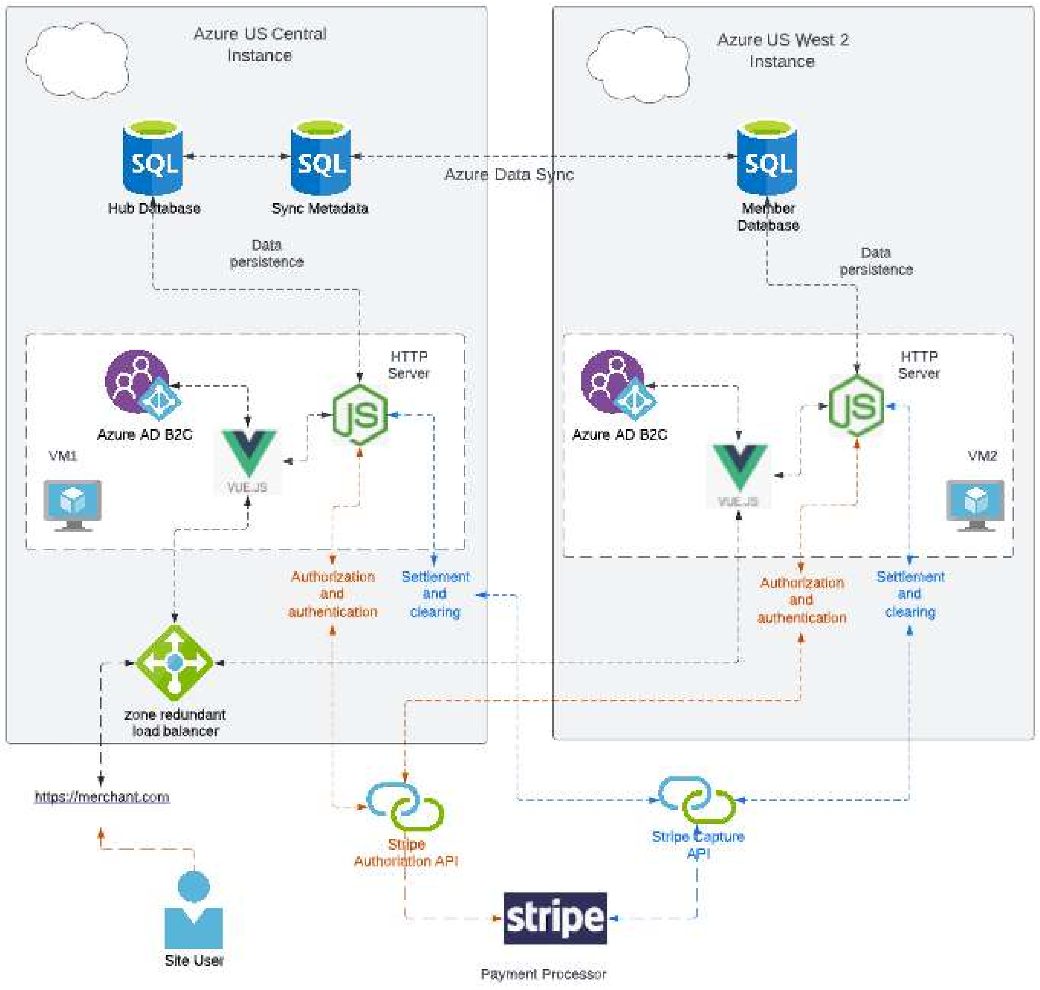
Director: Michael Batty, PhD

Dean: Michael J. McGuire, MLS

Contents

Introduction

This paper describes the software implementation for a distributed eCommerce web application integrated with a credit card payment processing system. The system is designed for the Azure cloud, implementing two load-balanced Virtual Machines and redundant Azure database instances. This application integrates with Stripe.com as credit card payment processor, and has Azure AD B2C for identity management. The system diagram is illustrated below:



Several elements of the design were not built in this proof-of-concept (POC), including:

- No Azure database instances were created
- Azure AD B2C was not implemented for identity management

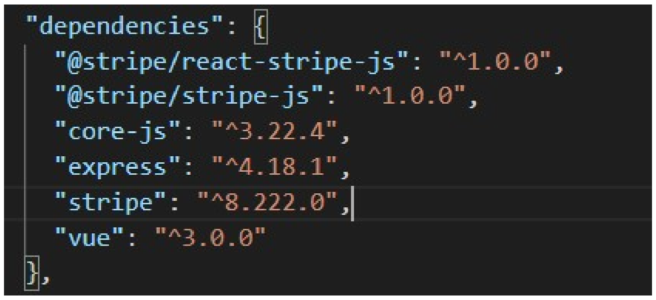
- The code was not deployed to Virtual Machines and the application is not currently load-balanced.

However, many elements were developed for this POC including:

- eCommerce website (Vue.js)
- HTTP proxy server (Node.js)
- Checkout page and Integration to Stripe.com
- Credit Card validation elements
- Fraud Detection implementation via Stripe Radar (Stripe, NDa)
- Success page confirming successful payment

How to Run the Application:

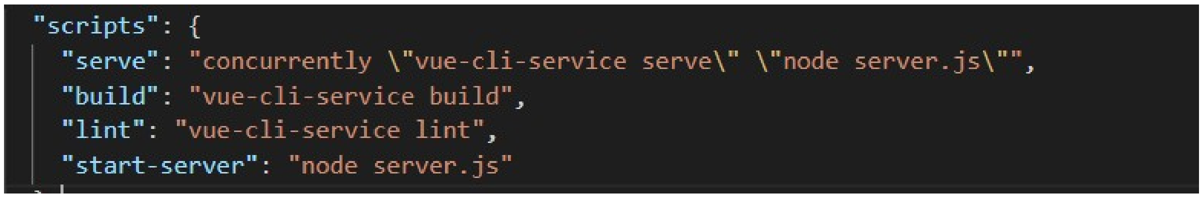
A folder with all the source code has been provided.   The application will require the installation of Node and NPM to handle package dependencies (npmjs.com, ND). An illustration of the dependencies that NPM will handle is shown below:



```
"dependencies": {
  "@stripe/react-stripe-js": "^1.0.0",
  "@stripe/stripe-js": "^1.0.0",
  "core-js": "^3.22.4",
  "express": "^4.18.1",
  "stripe": "^8.222.0",
  "vue": "^3.0.0"
},
```

Using VS Code is recommended to run/view the code, although running the application in other IDEs is possible.

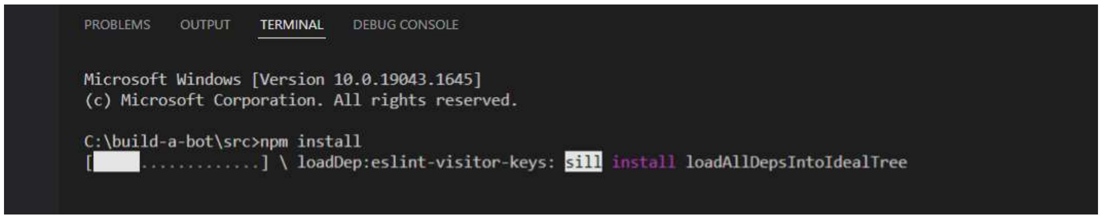
The code is configured to start both the Node.js server and the Vue.js web application by using “concurrently” in the package.json:



```
"scripts": {
  "serve": "concurrently \"vue-cli-service serve\" \"node server.js\"",
  "build": "vue-cli-service build",
  "lint": "vue-cli-service lint",
  "start-server": "node server.js"
```

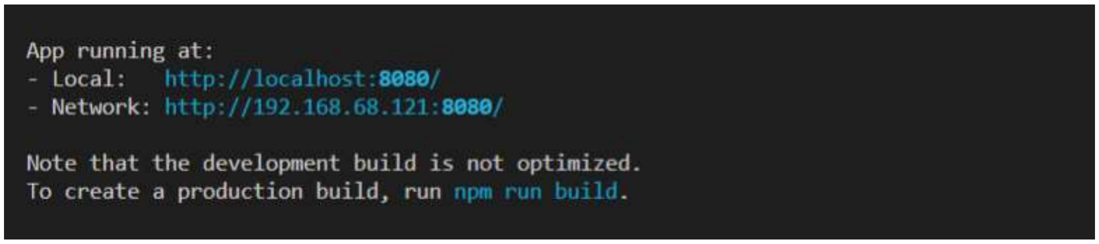
To run the application, open the folder in VS Code. Right click on the “src” folder, and select “Open in Integrated Terminal”.

Type *npm install* and then enter. This will install all the necessary dependencies. This should kick off some installs until it finishes:



```
Microsoft Windows [Version 10.0.19043.1645]
(c) Microsoft Corporation. All rights reserved.

C:\build-a-bot\src>npm install
[████.............] \ loadDep:eslint-visitor-keys: sill install loadAllDepsIntoIdealTree
```

Now you are ready to run the app. Type *npm run serve* and then enter to start the Node.js server and start the Vue.js application. When complete, it should look like:

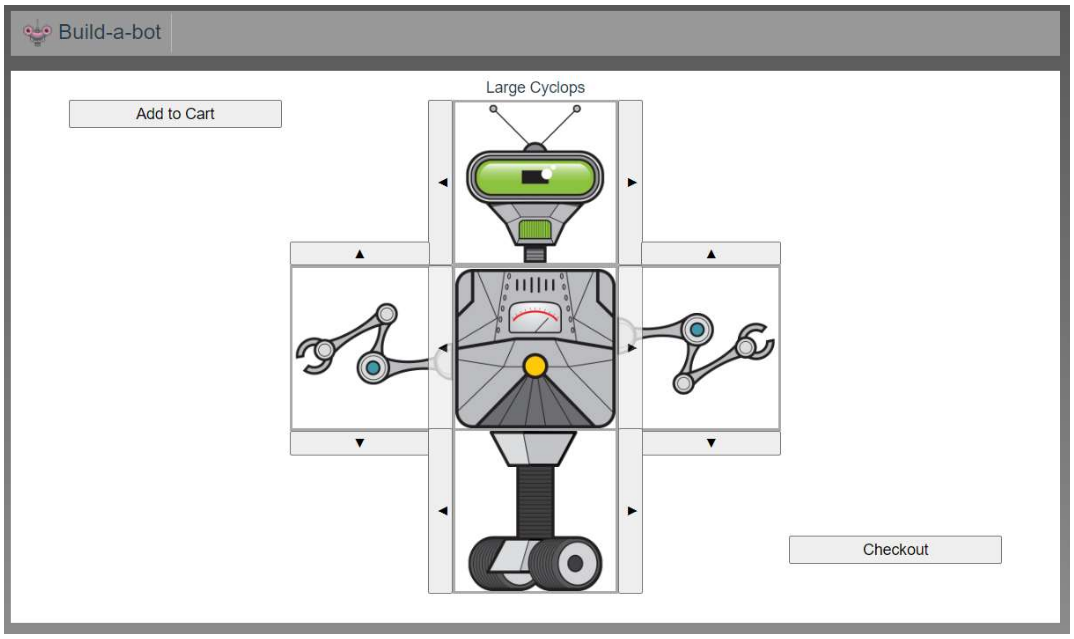


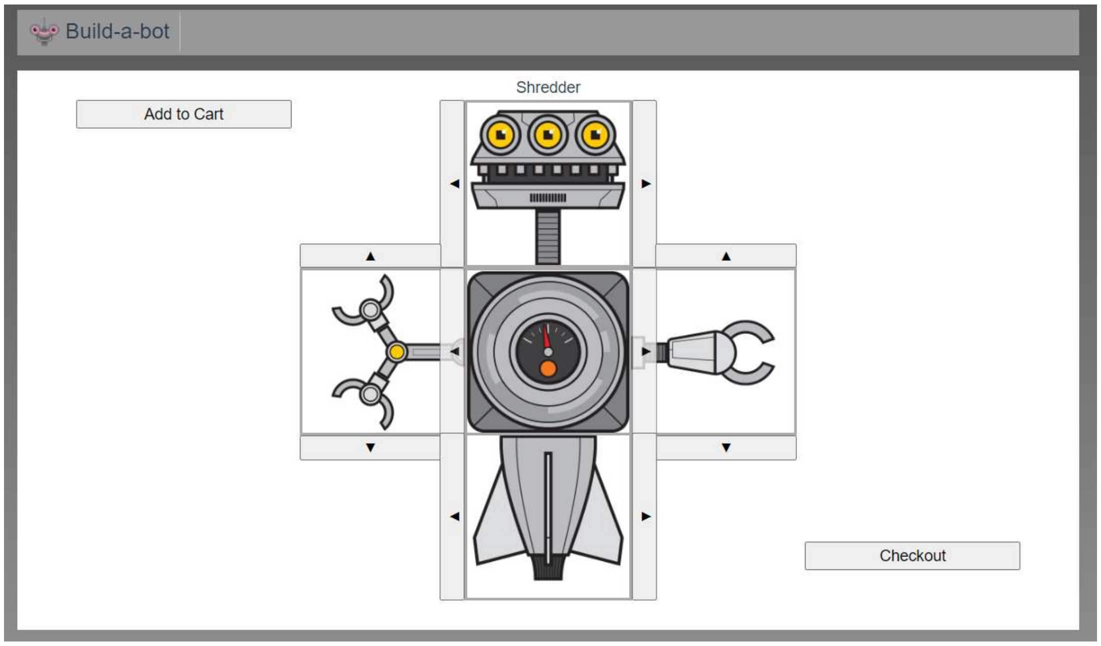
```
App running at:
- Local:   http://localhost:8080/
- Network: http://192.168.68.121:8080/

Note that the development build is not optimized.
To create a production build, run npm run build.
```

CTRL+ Click will then launch the app on localhost.

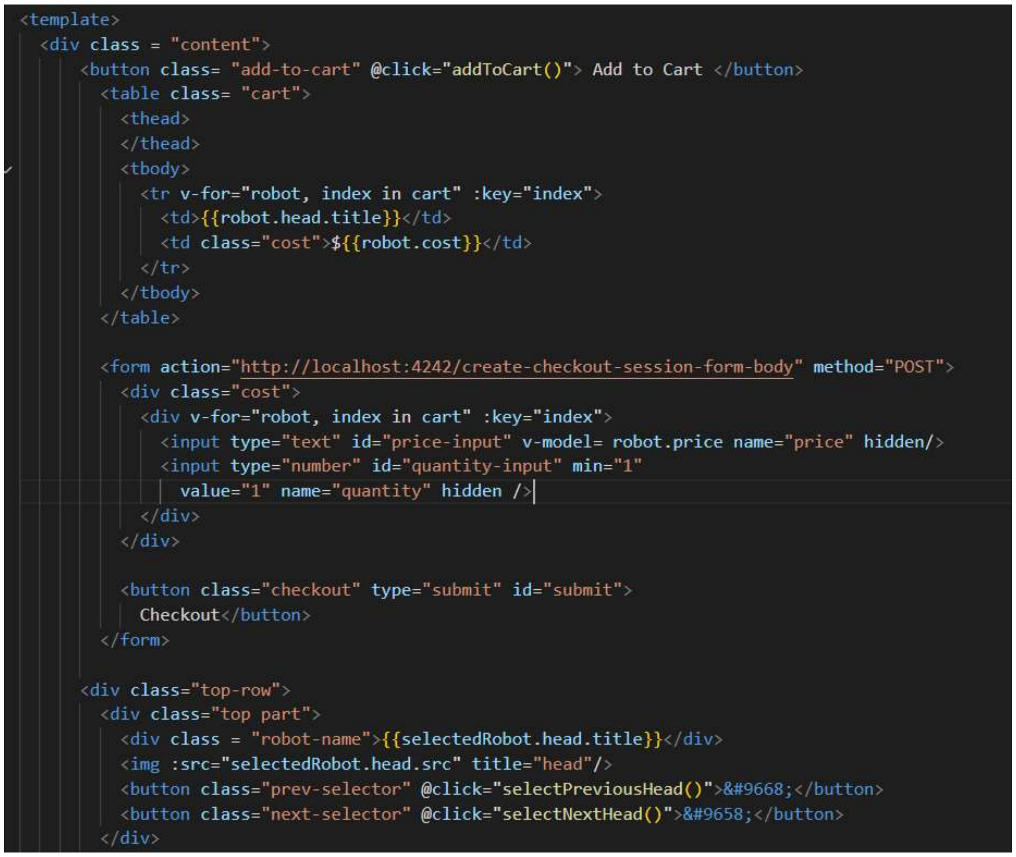
eCommerce Website (Vue.js)

The website is a Vue.js application called Build-a-bot. It is based primarily on a Pluralsight course, “*Vue.js Fundamentals*” by Jim Cooper (pluralsight.com, ND). The site’s home page allows a user to select various robot parts and build a robot of their choice. The Pluralsight course covered the “robot” elements, while I subsequently added the checkout flow via Node.js and Stripe.com.   The following screenshots show the application with some example robot types selected:



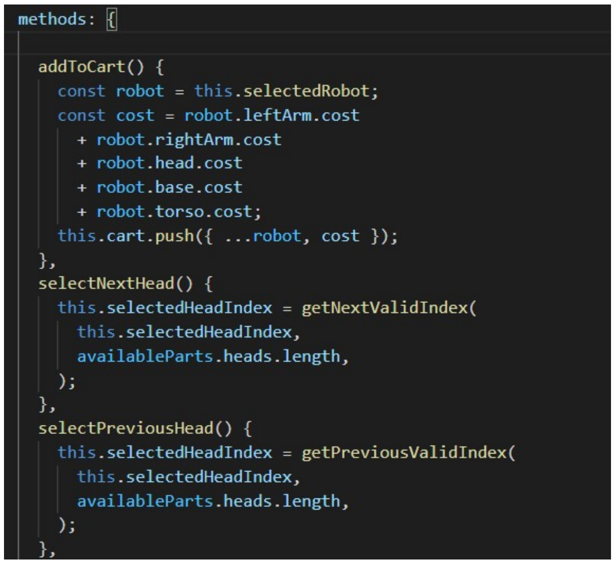


The following snippet shows some of the Vue "Template" section in the source code, which is where HTML elements are declared. The app utilizes an HTML Form POST method with a body that provides the details of the shopping cart to the Node server:



```
<template>
  <div class = "content">
      <button class= "add-to-cart" @click="addToCart()"> Add to Cart </button>
        <table class= "cart">
          <thead>
          </thead>
          <tbody>
            <tr v-for="robot, index in cart" :key="index">
              <td>{{robot.head.title}}</td>
              <td class="cost">${{robot.cost}}</td>
            </tr>
          </tbody>
        </table>

        <form action="http://localhost:4242/create-checkout-session-form-body" method="POST">
          <div class="cost">
            <div v-for="robot, index in cart" :key="index">
              <input type="text" id="price-input" v-model= robot.price name="price" hidden/>
              <input type="number" id="quantity-input" min="1"
                value="1" name="quantity" hidden />
            </div>
          </div>

          <button class="checkout" type="submit" id="submit">
            Checkout</button>
        </form>

      <div class="top-row">
        <div class="top part">
          <div class = "robot-name">{{selectedRobot.head.title}}</div>
          <img :src="selectedRobot.head.src" title="head"/>
          <button class="prev-selector" @click="selectPreviousHead()">&#9668;</button>
          <button class="next-selector" @click="selectNextHead()">&#9658;</button>
        </div>
```

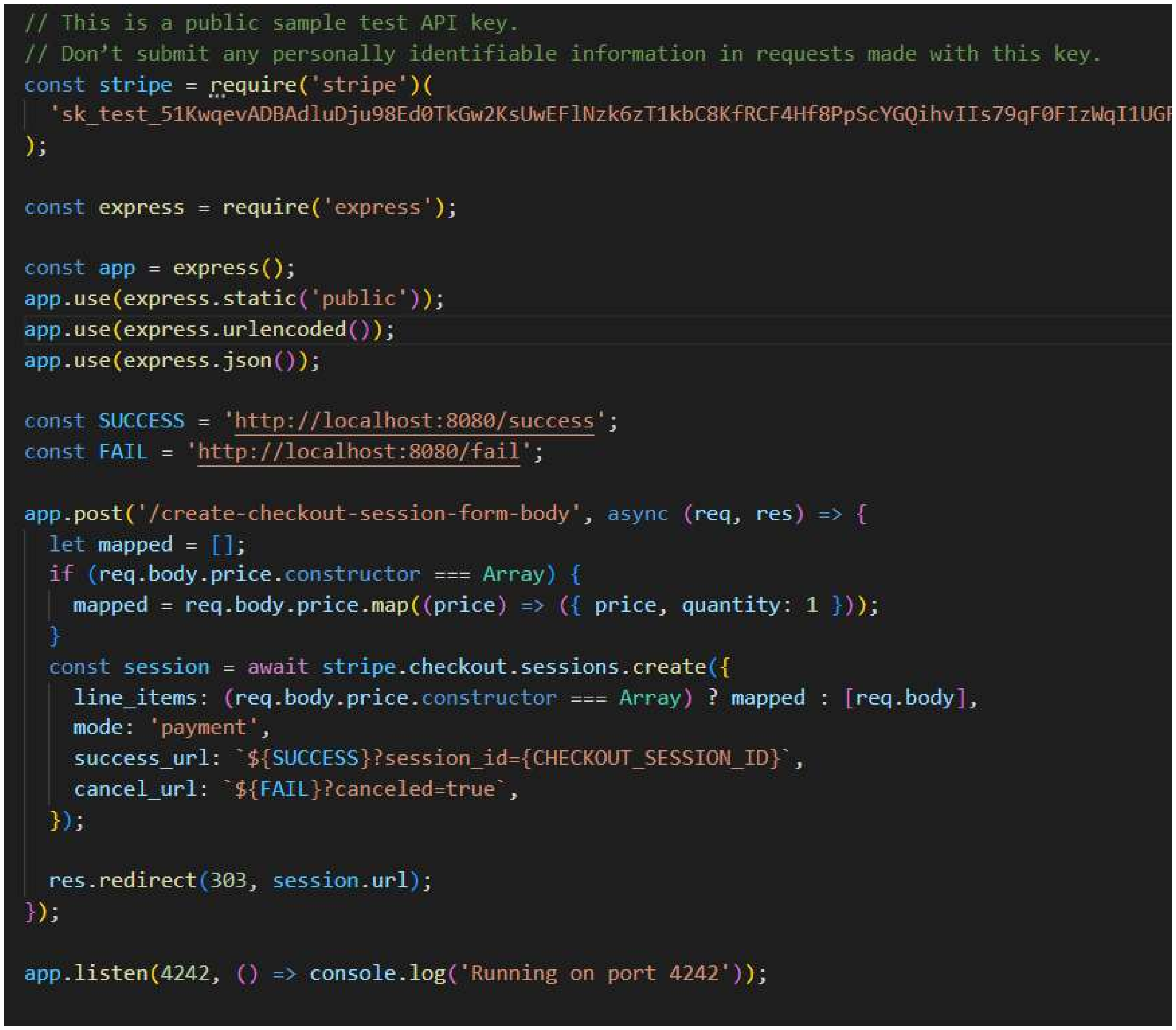
The snippet below shows some of the JS methods called from within the template to select the robot being purchased:



```
methods: {

  addToCart() {
    const robot = this.selectedRobot;
    const cost = robot.leftArm.cost
      + robot.rightArm.cost
      + robot.head.cost
      + robot.base.cost
      + robot.torso.cost;
    this.cart.push({ ...robot, cost });
  },
  selectNextHead() {
    this.selectedHeadIndex = getNextValidIndex(
      this.selectedHeadIndex,
      availableParts.heads.length,
    );
  },
  selectPreviousHead() {
    this.selectedHeadIndex = getPreviousValidIndex(
      this.selectedHeadIndex,
      availableParts.heads.length,
    );
  },
```

HTTP Proxy Server (Node.js)

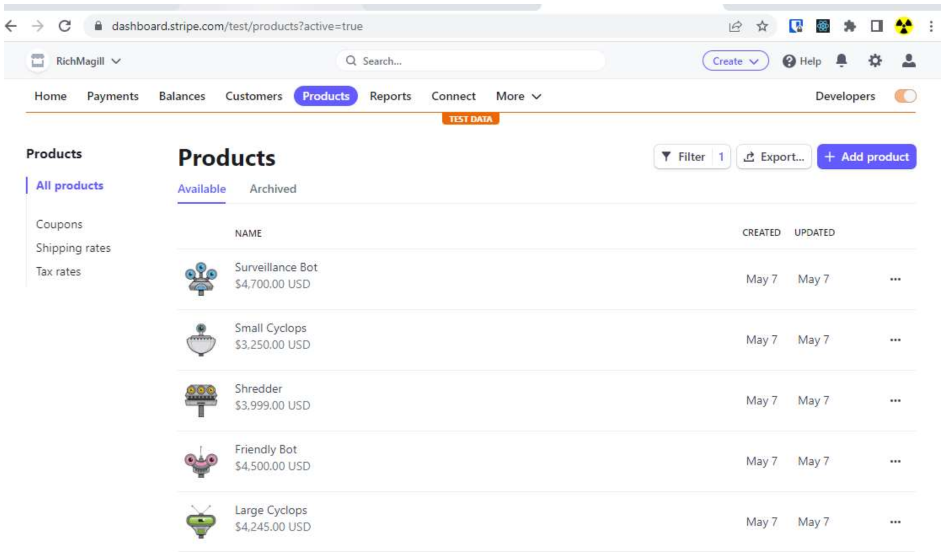
The Node server runs in the background, and is responsible to receive the Post from checkout and then formats the request body for Stripe. It then redirects the request to the Stripe.com URL. Note that a test "key" is required to implement Stripe. I got this by creating an account with Stripe. Upon posting to Stripe, the credit card information is processed (in TEST mode), and either a Success (200 OK) or Failure response is received. Upon receiving the response, the Node server then redirects to the appropriate Success or Cancel URL.



```
// This is a public sample test API key.
// Don't submit any personally identifiable information in requests made with this key.
const stripe = require('stripe')(
  'sk_test_51KwqevADBAdluDju98Ed0TkGw2KsUwEFlNzk6zT1kbC8KfRCF4Hf8PpScYGQihvIIs79qF0FIzWqI1UGF
);

const express = require('express');

const app = express();
app.use(express.static('public'));
app.use(express.urlencoded());
app.use(express.json());

const SUCCESS = 'http://localhost:8080/success';
const FAIL = 'http://localhost:8080/fail';

app.post('/create-checkout-session-form-body', async (req, res) => {
  let mapped = [];
  if (req.body.price.constructor === Array) {
    mapped = req.body.price.map((price) => ({ price, quantity: 1 }));
  }
  const session = await stripe.checkout.sessions.create({
    line_items: (req.body.price.constructor === Array) ? mapped : [req.body],
    mode: 'payment',
    success_url: `${SUCCESS}?session_id={CHECKOUT_SESSION_ID}`,
    cancel_url: `${FAIL}?canceled=true`,
  });

  res.redirect(303, session.url);
});

app.listen(4242, () => console.log('Running on port 4242'));
```

Stripe Integration

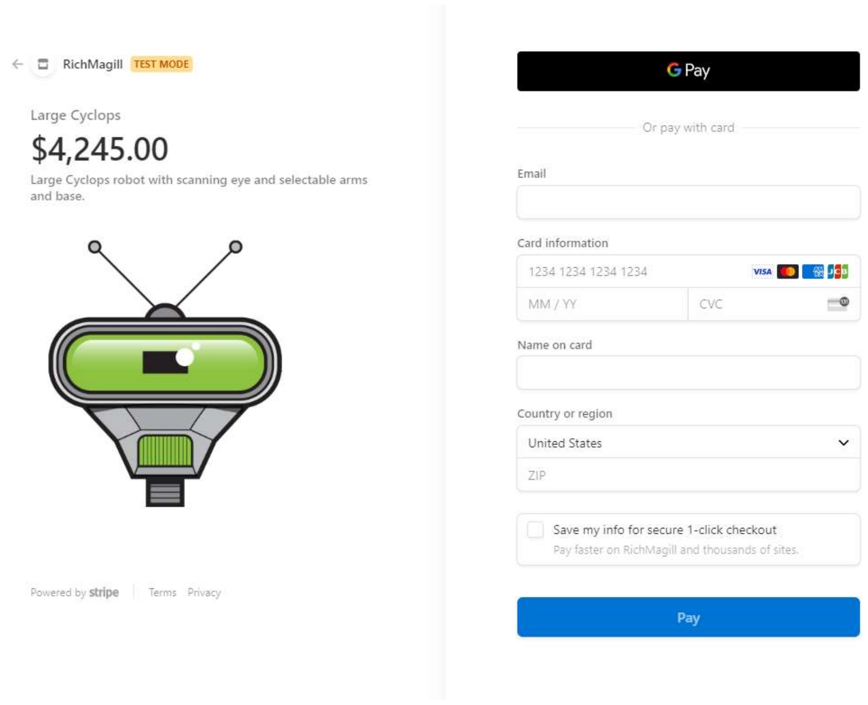
As mentioned previously, I created an account in the Stripe application. The next step was to enter Products and Prices in the stripe dashboard, including all the various robot types (Stripe, NDb).



Products

NAME	CREATED	UPDATED
Surveillance Bot $4,700.00 USD	May 7	May 7
Small Cyclops $3,250.00 USD	May 7	May 7
Shredder $3,999.00 USD	May 7	May 7
Friendly Bot $4,500.00 USD	May 7	May 7
Large Cyclops $4,245.00 USD	May 7	May 7

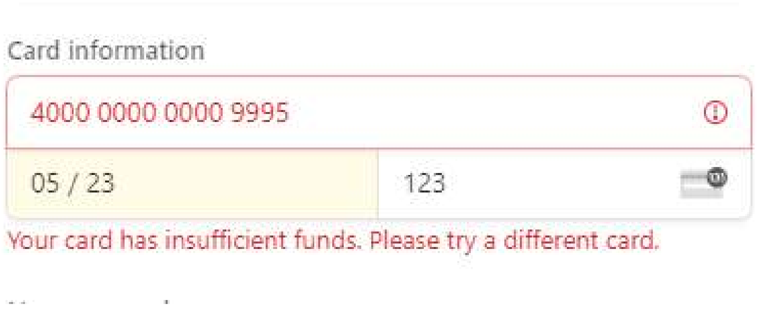
With the Node server, test key, and products in place, the user is then presented with a Stripe.com checkout page when they select a robot and click "checkout":



Large Cyclops
$4,245.00
Large Cyclops robot with scanning eye and selectable arms and base.

Testing and Response

Stripe provides many different fake credit card numbers to test scenarios that may be encountered when processing credit cards.   Here is a card with insufficient funds:



Card information

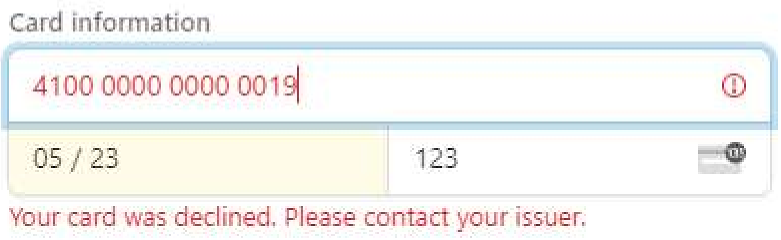
4000 0000 0000 9995

05 / 23

123

Your card has insufficient funds. Please try a different card.

This card has been flagged for fraud:



Card information

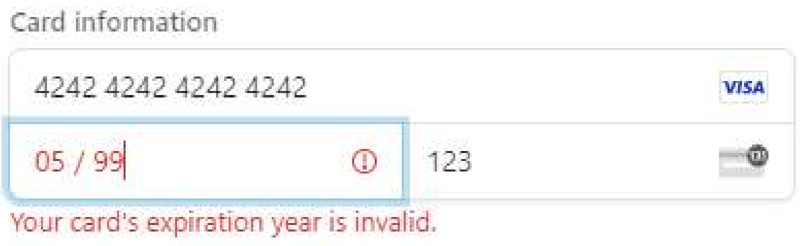
4100 0000 0000 0019

05 / 23

123

Your card was declined. Please contact your issuer.

This entry has an invalid date entered:



Card information

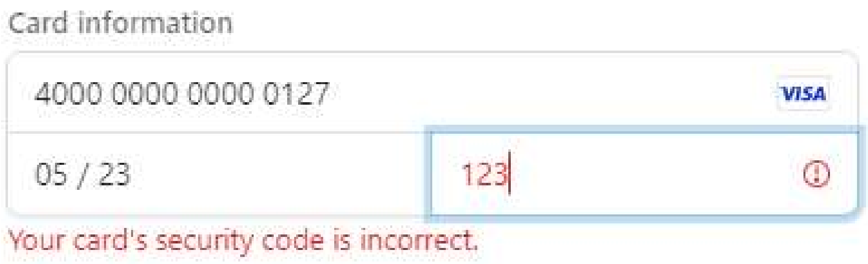
4242 4242 4242 4242

05 / 99

123

Your card's expiration year is invalid.

This is an example card with the wrong CVC entered:



Card information

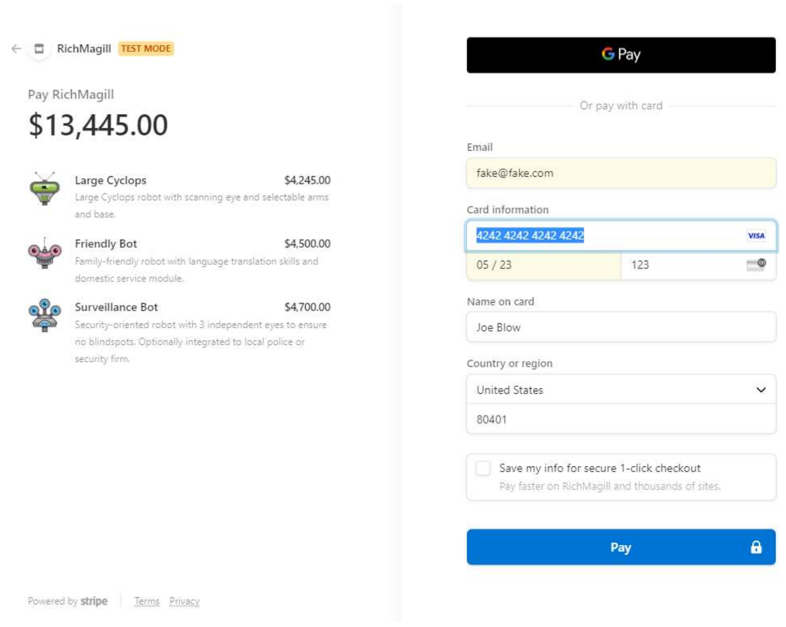
4000 0000 0000 0127

05 / 23

123

Your card's security code is incorrect.

Finally, here is a good test credit card number (4242 4242 4242 4242) with no fraud and sufficient funds.


← RichMagill TEST MODE

Pay RichMagill

$13,445.00

Large Cyclops Large Cyclops robot with scanning eye and selectable arms and base.	$4,245.00
Friendly Bot Family-friendly robot with language translation skills and domestic service module.	$4,500.00
Surveillance Bot Security-oriented robot with 3 independent eyes to ensure no blindspots. Optionally integrated to local police or security firm.	$4,700.00

Powered by stripe | Terms Privacy

G Pay

Or pay with card

Email: fake@fake.com

Card information: 4242 4242 4242 4242 VISA
05 / 23 123

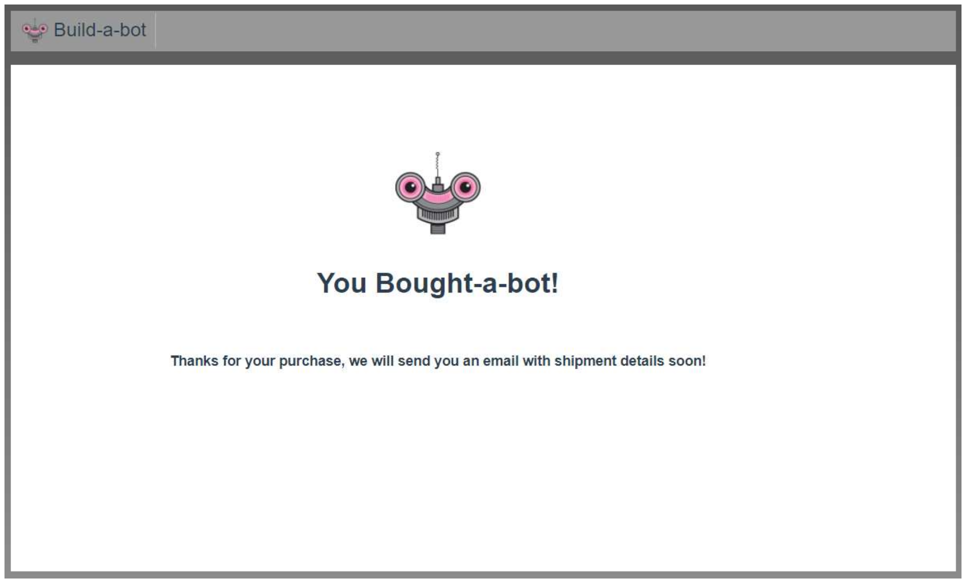
Name on card: Joe Blow

Country or region: United States
80401

Save my info for secure 1-click checkout
Pay faster on RichMagill and thousands of sites.

Pay

Upon success (receiving 200 OK from Stripe), the Node server redirects the user to the success page:


Build-a-bot

You Bought-a-bot!

Thanks for your purchase, we will send you an email with shipment details soon!

Conclusion

The application was successfully implemented with a Client (Vue) and Server (Node) model that integrates to the Stripe.com API and processing application. The functionality in that respect is very high. Stripe implementation ensures PCI compliance (Stripe, NDc), and further provides styling, validation, error-handling, Luhn algorithm checking, and even fraud detection via Stripe's built-in Radar application. The code is currently implemented with a "Test" key, and to actually go-live all I would need to do is to replace the test key with a "Production" key.

That said, there is still significant work required to fully implement the design in the cloud with redundant instances of application and database:

- Database instances are still required. The database will allow for the application to track user interaction, orders, products, and other critical persisted data. The Node server would eventually be linked to the database and would provide HTTP endpoints to the client for these connections. Database replication would also need to be implemented via Azure Data Sync.
- Azure Active Directory B2C still needs to be setup to handle Identity management. In the current POC, all transactions are being performed without user accounts and without the application having any knowledge of the user identity. Only at the point that the credit card is transmitted to Stripe does the user present any identifying information. An important next step is to secure the application with appropriate Authentication and Authorization controls.
- The application is not yet deployed to an actual VM, and is not load balanced. In order to do this, Azure Resource Manager (ARM) templates need to be developed. ARM templates are JSON templates that describe all the elements in the environment, and

can then rapidly deploy the resources (VMs, Load balancer, etc.) from the Azure CLI (Microsoft 2022). Multiple templates can be developed to allow for rapid scaling.

- Finally, in order to maintain and deploy this application professionally, we will need to establish appropriate environments (DEV, TEST, STAGE, PROD) and Continuous Integration and Continuous Development (CICD) pipelines to keep the environments updated with the latest code.

While there is still a lot to do to get the application in the cloud and in production, the POC was successful and demonstrates a really nice integration between Vue, Node, and Stripe.

References

Stripe, NDa. *Radar. Fight Fraud with the strength of the Stripe network.* Retrieved May 11, 2022 from https://stripe.com/docs/radar

npmjs.com, ND. *Downloading and installing Node.js and npm.* Retrieved May 11, 2022 from https://docs.npmjs.com/downloading-and-installing-node-js-and-npm

pluralsight.com, ND. *Vue.js Fundamentals by Jim Cooper*. Retrieved May 11, 2022 from https://app.pluralsight.com/library/courses/vuejs-fundamentals/table-of-contents

Stripe, NDb. *Manage products and prices.* Retrieved May 11, 2022 from https://stripe.com/docs/products-prices/manage-prices

Stripe, NDc. *A Guide to PCI Compliance*. Retrieved May 11, 2022 from https://stripe.com/guides/pci-compliance

Microsoft, 2022. What are ARM templates. Retrieved May 11, 2022 from https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/overview