Final Report for CPSC490: Special Projects, Spring 2018

André Monteiro

Advisor: Ruzica Piskac

SMS CRM: A Solution Facilitating the Connection Between Consumer and Company

Customer Side: monteiro-senior-project.herokuapp.com

Company Side: monteiro-senior-project.herokuapp.com/dashboard

Login Info: username/password — ruzica/test

Abstract

In the landscape of e-commerce companies, oftentimes the struggle comes in facilitating interactions between businesses and customers and incentivizing reorders. One of the prominent mechanisms for this communication has traditionally been e-mail. The struggle with e-mail in recent years has been the sheer quantity of e-mails that end users are receiving and sorting through on a daily basis. E-mail marketing has turned into a problem rather than a solution. A way to cut through the noise is communicating with customers via SMS, and the ability to process re-orders over text can be leveraged to make it easier for customers to reorder. This also presents an opportunity for companies to deliver their brand to customers in an extremely targeted way. The goal of this project is thus to create a simple e-commerce store that funnels customers into an SMS CRM (customer relationship management) system that allows for two-way communication between customers and businesses. The e-commerce
store will be the customer-facing component that allows them to learn about and purchase some hypothetical product. The CRM will allow the business to talk to customers and do things such as flag certain customers, place new orders for them, edit their shipping information, etc. In addition to this, team members of the e-commerce company will have access to a private dashboard accessible via a login portal that will display key performance indicators, graphs measuring company growth, tables displaying orders and customers, and display the actual CRM that will allow for dialogue with customers. By leveraging natural language processing, sentiment analysis can be performed on incoming SMS messages from customers to determine whether a text is positive, negative, or neutral so that a company can have a better understanding of the emotions of their customers.

**Background**

The idea for this project came from a few different places. In particular, I’ve been interested for a while in the applications of technology in the context and at the intersection of how the Internet has changed mechanisms for delivering consumer products. It is clear that the Internet has had a tremendous impact on delivering tech solutions to end-consumers, but for consumer products, and specifically for consumer packaged goods, it is only in the last few years that companies have begun to leverage the power of the Internet. Examples of impressive direct-to-consumer brands include companies like Casper, Allbirds, Daily Harvest, and many more. The ways in which
these companies communicate value propositions to customers and try to convince a
customer to make a purchase are completely revolutionized by the landscape in which
these interactions occur. I’ve been curious whether or not there is a more powerful way
to communicate with customers by leveraging technology. From this question arose the
idea of an e-commerce solution that allows customers to be placed into a texting
funnel wherein they can communicate directly with the team behind a company and
place reorders over text, reducing friction and allowing for a completely branded
experience.

In addition to this, I wanted to work on a full-stack project that would allow me
to hone skills and practice coding on a larger scale than is typical of a class project. I
wanted, in particular, to connect multiple technologies and APIs to really push the
boundaries of what the project could do. Being able to stand upon and leverage the
work of other developers is a crucial part of the development process, and I wanted to
ensure that the project I was building did not exist in isolation but rather worked within
an ecosystem of existing code.

**Screenshots/Walkthrough**

There are two main components to this web-app. The first is a customer-facing side,
which is a simple e-commerce site wherein customers can purchase a product online.
As a test product, this site sells hand sanitizer in a pack option and a trial option.
In order to practice the front-end component, the customer-facing side also has a mobile view option, which allows customers to browse the site on their mobile phones with a more friendly user experience. The front-end of the website was done by leveraging Bootstrap to ensure consistency in the layout, and simple CSS tweaks to make it a little bit more aesthetically pleasing. On clicking one of the order buttons, a customer is brought to the cart page, where they can fill in their information to pay for the product and have it shipped to them. This page also has a mobile view to allow users to check out easily on their phones. The cart stores user sessions to allow for a
more seamless and frictionless checkout experience. This is done by leveraging an express module that stores sessions based on requests to make them stateful rather than stateless. A customer can check out, in this case using a test Stripe card (card number 4242 4242 4242 4242, any expiration date in the future, any CVC, and any zip code). The cart also handles card errors by utilizing Stripe’s built-in error-handling code, which can handle things from invalid card numbers to invalid expiration dates, and many more errors that may arise when inputting card information online. Once an order is placed, a notification is sent using
the Slack API to a Slack channel where the company's team members can be alerted about the new order.
At this point, team members can go to the dashboard and access important information about customer’s orders and see metrics that track key performance indicators. Here, the team member is greeted by a login page. This is done in order to protect private customer information from being accessed by people outside of the company. Once logged in, the team member is brought to the dashboard, which displays key metrics to track how the company is doing. These metrics are discovered by performing queries on the database and passing the variables into an EJS template in order to display them on the client-side. These queries are all in the file ./routes/dashboard.js. There are also views for displaying the information stored in the tables. The schema for these tables can be found in the ./texttobuy/models/customer.js file. The orders page lists out all the orders in the table, organized in an easy-to-read fashion and giving accessible links to both the
individual order page and
to the customer's profile in
the CRM. The CRM
organizes customers
based on recency of
conversation and whether
or not text messages
have been read by a
team member.
Customers are also
able to be flagged to
help team members
remember to return to
conversations.
Unread messages are
tagged with a blue
dot to better help
team members find
conversations to
reply to, but can also
be accessed via the button on the left sidebar. Flagged conversations show up when
the button on the sidebar is clicked. Clicking on a chat preview on the left sidebar
brings up the main CRM view, where the conversation is filled in on the middle section
and the customer information is piped into the right sidebar. The conversation section
does many things. It displays texts back and forth between the customer and the
company and styles them appropriately for a good user experience. It detects whether
a customer’s text message is an image and displays it appropriately. Messages each
have a timestamp associated with them that can be displayed by clicking on the
message. Messages from customers also each have a sentiment associated with them,
either positive, negative, or neutral. Positive messages are displayed in green and
negative messages are displayed in red. Scrolling up in the conversation dynamically
loads in older messages. The right sidebar is where many actions can occur — here the employee can place an order for a customer when the customer asks for more products. The customer's personal and shipping information can be edited, customer data can be displayed, and there are buttons marking a customer to be flagged or added to a no-text list. When a customer is added to the no-text list, the next time an employee clicks on their conversation, the middle section becomes grayed out and they cannot send messages to the customer. Whenever a customer texts the phone number, the new message is displayed appropriately at the top of the conversation list and, in the case that an employee is already on the customer's conversation, the new message automatically gets appended to the conversation section using a socket.io emit event. In addition to this, when two employees are on the same conversation with a customer, their states are emitted to all other client connections. This allows for the emission of events such as alerting other clients when a user is typing. This helps to prevent conflicts among employees by ensuring that all connected users are aware...
when a team member is about to send a message to a customer so that team members do not send superfluous messages. These pictures also show the mobile-optimized view of the conversation section to allow for employees to respond while on-the-go. As seen below, the left and right sidebars also have mobile-optimized views.
Technical Details

The web-app itself is hosted on Heroku with automatic deploys from the master branch of a Github repository. Heroku was a good option because it offered free hosting on its hobby plan and easily scales up without needing to change anything about the implementation.

With the rising popularity of no-SQL databases in recent years, I decided to try my hand at working with MongoDB since I’d already done another project in SQL. The database is hosted on mLab, a cloud database that works easily with Node modules that connect to remote databases. MLab also offers a free option up to a certain database size so it was an easy winner.

Payment processing was done using Stripe, which offers a testing mode where a fake card can be used to process payments. The cart implements an instance of Stripe elements, which allows for the tokenization of credit and debit cards to facilitate secure transactions between e-commerce sites and Stripe’s servers. In addition to the Stripe side of things, I protected the web-app from bad requests by only passing back the ID and quantity of the products being ordered. Pricing of the products was handled fresh on the backend in order to prevent malicious users from deducing the request format to charge themselves lower prices at checkout.

Text messages were handled using a Twilio phone. When a customer sends a message to the phone number, a web hook is sent to a POST route on the web-app’s
server and the message is handled appropriately from there, being associated with the correct customer or creating a new customer on first-contact. If the message is a picture, that gets tagged appropriately to display within the CRM. The sentiment analysis also occurs at this point, with the body of the message being read and the positive, negative, or neutral tag being assigned just before the text gets added to the database. When a text message needs to be sent to the customer, the Twilio node module is used, although alternatively, a POST request could be sent to Twilio using the appropriate authentication method. These messages are inherently stateless, so they need to be stored in a format that emulates the flow of dialogue, in this case as a subset of each individual customer, where each customer is associated with a phone number.

The project was done in Node.js and uses existing modules that perform tasks such as sentiment analysis and APIs such as Stripe for payment processing. To protect sensitive information such as the database URL and secret API keys, the app uses environment variables to avoid exposing these keys. Important modules include express for routing the web-app, ejs for templating HTML files to allow for the passing of variables from server-side to client-side, mongoose for connecting to the database and performing CRUD operations and queries, node-sentiment for performing sentiment analysis on text messages from customers, passport for authenticating login attempts on the dashboard, socket.io for the emission of events to dynamically update
the client-side experience for team members on the CRM, stripe for payment processing, twilio for handling text message processing, and underscore to allow for the delay of function calls until various asynchronous functions were done running. The @slack/client module had some issues so I ended up using POST requests to push notifications to the Slack channel. These modules were crucial in allowing me to spend the majority of my time focusing on more important technical challenges rather than being bogged down in the details that didn’t affect the higher-level scope of the project.

Future Possibilities

There are many possibilities for next steps on this project. First and foremost would be to add more products to the e-commerce site. As it stands, there are two iterations of one product, but these have been sort of hard-coded. In order to expand the scope of the company, the web-app could incorporate a CMS (content management software) component to allow non-technical team members to add, edit, and delete things like products, promotion codes, etc. Another clear step would be separating the backend code from the e-commerce site to which it is attached. This would be the first step in packaging the software into a unit that can then be sold to other e-commerce companies as a SaaS product. Other companies that want to be able to text with their customers and allow customers to place orders over text would
connect with this product to handle the cart section of their website (similar to options like Shopify or Snipcart) and then be given access to a backend dashboard that is unique to their company to be able to track their metrics. In addition to these options, a heavier NLP and AI approach can be taken with the dialogue between customer and company. The short-lived attempt to create a pseudo-chatbot that processed orders automatically resulted in a lackluster user experience, and creating a more intelligent processing agent, especially in the context of a client that sells multiple products, would be an entire project in and of itself.

Acknowledgements

Thank you to Professor Piskac for her guidance on this project and thank you to the Yale Computer Science Department for teaching me the foundations of Computer Science without which I would never have been able to accomplish a project of this scope. I also want to acknowledge the kind souls who reply to Stack Overflow posts from 2012 with one upvote — you will never know how much you mean to me but I think of you always.