Customer Driver Communications Service

OrangeWorks FA21 Project Report

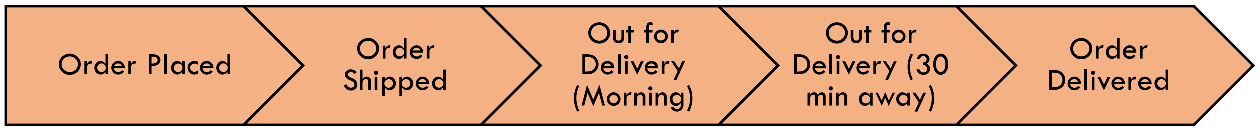
Rohan Maheshwari & Jenny Gibson

Project Overview

The Customer Driver Communications Service (CDCS) is an API developed to allow delivery drivers to communicate with customers for the duration of a delivery. There are several features such as anonymous messaging, simultaneous active conversations, saving chat logs, and a status dashboard that enable the functionalities of this service. We introduce this product to reduce complaint calls to THD call centers regarding logistical issues in delivering orders to customers.

Background and Problem Space

Home Depot delivery starts with its tracking system. From the placement of the order to its delivery, the customer can always see the status of where their order is on that timeline. On the morning of the delivery, the customer generally receives a text with a message informing them that their order will be delivered today. Then, 30 minutes before the order will arrive, they are notified again that their order is 30 minutes away. For special deliveries that may require installation or some other procedure, customers are also notified with a 4-hour window in which their delivery will be made. While this is certainly a functional system that can suffice for most customers, there are still many cases in which this model may not work. Research in Customer Support has shown that almost 5% of all deliveries end with a call to customer support to express an inconvenience with their delivery. For example, many customers living in apartment complexes experience delays in their delivery because the driver does not know the gate code to get in. In other cases, a driver may be held up at another customer's home for a delivery, and thus the next customer experiences delays. Like these cases, there are many more situations that cause logistical issues in deliveries that occur due an issue on either the customer's end or the driver's end, or even both.



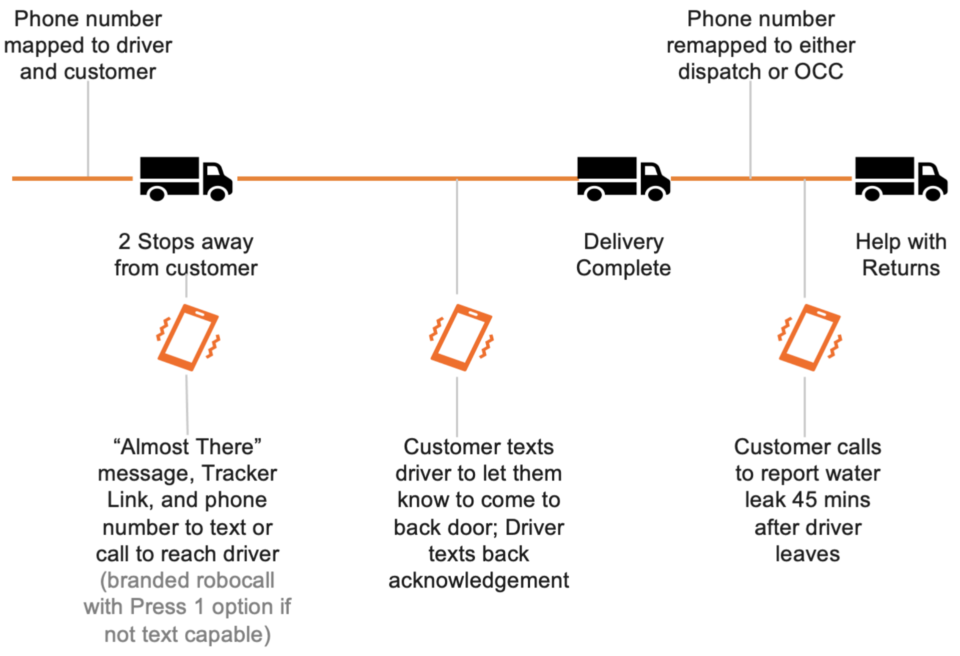
Order Tracking Flow (Customer Perspective)

Solution Proposal

To mitigate the issues that can be caused by unexpected logistical issues, we believe that communication is the key. For small issues such as a missing gate code, the customer can quickly communicate to the driver that necessary information. For situations where the customer is running late for a delivery, they needed to be present for, they can give alternative instructions to the driver on where to leave the order. Introducing a line of communication can resolve many small-scale issues that would currently prevent entire deliveries from not happening.

However, many times, an installation may take longer than necessary at one customer's house which will inevitably delay the delivery of the next person. Situations like these are hard to prevent, but perhaps being able to relay the status of the current situation to a necessary party can reduce the complaint calls made to the call center. After all, the goal is kept customers satisfied with the delivery service, and growth in that aspect can be measured by reduced complaints.

Therefore, we propose a communication platform that allows customers and drivers to communicate for the duration of their delivery. This platform will allow customers and drivers to communicate anonymously via messaging and/or calling. We believe that introducing this platform can fulfill the situations mentioned above, and reduce the complaint calls to the call center.


Phone number mapped to driver and customer
Phone number remapped to either dispatch or OCC
2 Stops away from customer
Delivery Complete
Help with Returns
"Almost There" message, Tracker Link, and phone number to text or call to reach driver (branded robocall with Press 1 option if not text capable)
Customer texts driver to let them know to come to back door; Driver texts back acknowledgement
Customer calls to report water leak 45 mins after driver leaves

Delivery Flow with CDCS

Development Goals and Specifications

After the 3-week design sprint, we finalized the development roadmap which included the major goals, features, and technology we will use and implement to create the CDCS.

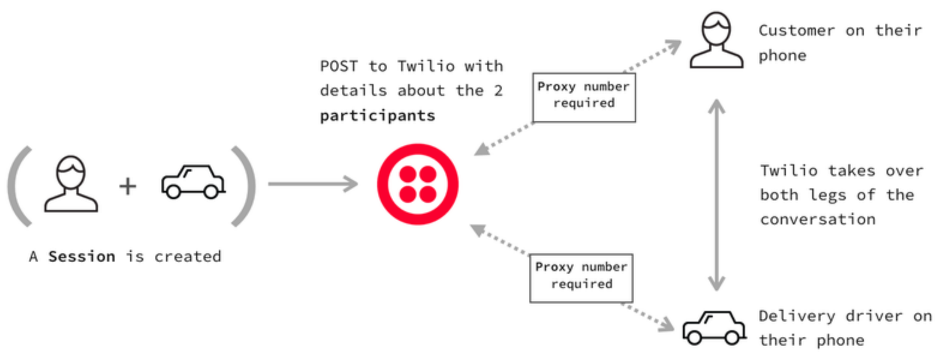
Goals

The biggest and most important goal of the semester was to develop the backend infrastructure to be able to support the platform. This included setting up a server, database, and client that would allow us as engineers to test the platform in real time and host the communication sessions between drivers and customers. Putting these three components together would result in the final product, an API service that would allow the current order tracking services to use to initiate customer-driver communication. Following this, we also wanted to create a simple dashboard that would allow delivery supervisors and customer support associates to be able to view the communication logs between drivers and customers to settle any issues or disputes regarding a delivery after it has occurred.

Technical Dependencies

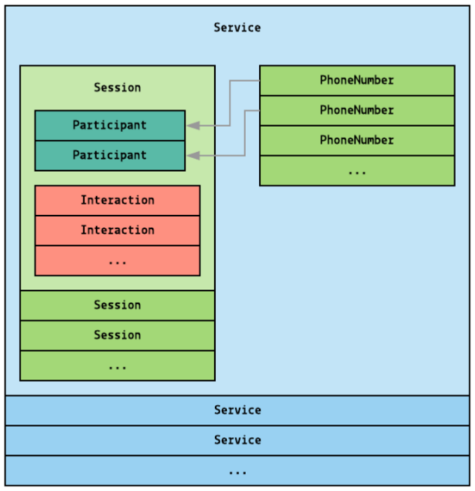
The CDCS has three main frameworks that are used in its development.

First we use Twilio's proxy conversation client to power our anonymous messaging and calling feature. Twilio uses special proxy numbers that allow two parties to connect without sharing their actual phone numbers.


POST to Twilio with details about the 2 participants
A Session is created
Proxy number required
Proxy number required
Customer on their phone
Twilio takes over both legs of the conversation
Delivery driver on their phone

Twilio Proxy Logic

Twilio uses the concepts of services, sessions, participants, and interactions in the functionality of their API. First, services are the highest level of all the resources and tools that are provided with Twilio's proxy API. It contains sessions and proxy numbers. Then, we have sessions that are the actual conversations between two participants. They contain the phone numbers of both participants as well as the Twilio proxy number that is used to mask the communication. Finally, there are the participants who are individuals with phone numbers who are connected to a session. For now, this level of abstraction will suffice for the scope of this project. More information and specifics about the Twilio Proxy API can be found here: https://www.twilio.com/docs/proxy



Twilio API Hierarchy

In the CDCS, we create services that are named after the driver and sessions that are named after the order id. This is because one service can have multiple sessions, just like a driver can have multiple conversations with customers. Since there is one conversation for each order, we maintain the uniqueness of the session id by giving it the same name as the order id.

Secondly, we use Firebase Realtime Database. This framework is relatively popular and used frequently, so we will keep this section brief. Firebase is a Cloud based platform that allows developers to host apps and store data. For the CDCS, we use Firebase's Realtime Database to store many types of data. Some of the data types that we store are:

1. Order Delivery Information (customer data, driver data, order number)
2. Twilio Services (driver name, Twilio phone number, service id)
3. Twilio Sessions (driver number, customer number, session id)
4. Chat Logs from Sessions (messages and their senders)
5. Twilio Proxy Numbers (usage status)

Lastly, we use Express.js in tandem with Node to set up our backend server. With Express, we create an API that allows clients and other services to interact with the proxy and firebase functions. Based on the updates from the THD order tracker, these API endpoints can be triggered to start the process of creating the proxy conversation between the driver and customer and end the process once the order has been delivered.

CDCS Design

To explain how the CDCS works, we will first discuss and explain the endpoints from the API we provide.

The following endpoints are unused in the backend but are provided for the developer to be able to manipulate the database manually for testing and simulation.

/writeOrderData → allows client to pass in order information (type: OrderWithID) as a request and post the information to the database.

/getOrderData → allows client to retrieve order information by passing in an order id as a request. Response is given as type Order.

/getAllOrderData →allows client to retrieve information for all order. Response is given as type Order[].

/writeTwilioNumber → allows client to pass a Twilio proxy number as a request and post the information to the database.

/writeServiceData → allows client to pass Twilio service data from a proxy as a request (and post the information to the database.

The following endpoints are the bread and butter of the CDCS. They are to be used in tandem with the THD tracker to create/delete conversations between drivers and customers, and to save the chat logs of those conversations.

/generateProxy → allows client to create a conversation between a driver and a customer by passing in an order id and order information as a request. Will also send an initial text to the customer. Requires a service to already exist for the driver. Returns an object indicating the success of the request, and existence of a session, and error object.

/checkService → allows client to check if a service exists for a driver by passing an order id. A service is necessary for a session to be created between a driver and a customer. If a service exists, a session will be created between the driver and the customer along with an initial text send to the customer. If a service does not exist, one will be created, but no session will be created afterwards. The client will have to call

/generateProxy → Returns an object indicating the success of the request, existence of a session, and error object.

/deleteProxy → allows client to delete a session between a driver and customer by passing in an order id.

/writeSessionLogs → allows client to write the logs of a session to the database by passing in an order id. Requires that the session is currently active. Returns an object indicating the success of the request and an error object.

/getSessionLogs → allows client to retrieve logs of a session from the database by passing in an order id. Does not require the session to be active. Returns an object containing the session log data.

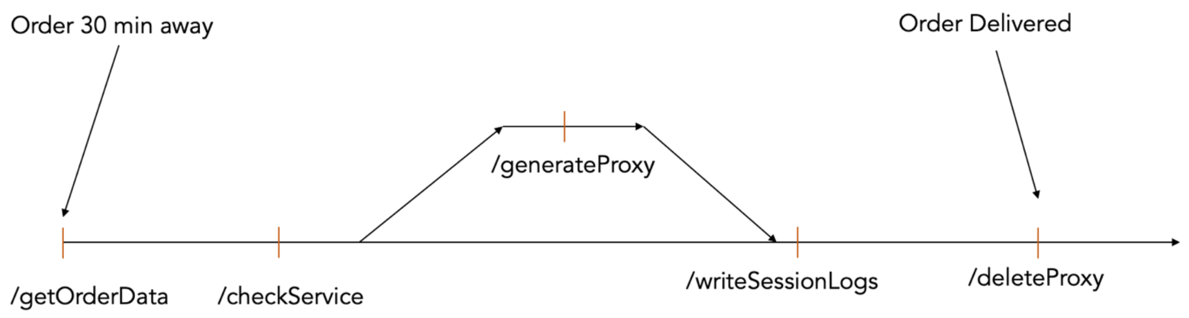
Outside of these endpoints, there many helper functions used internally to make each endpoint functional. We will not cover those here, as their implementation is trivial and self-explanatory.

API Event Flow

Moving into the final and most important part of the platform, we will describe the life cycle of a Twilio conversation.

When an order is 30 minutes from delivery, CDCS should trigger /getOrderData to retrieve the customer and delivery driver information related to that order. Then, the CDCS should trigger the /checkService endpoint. Based on the response, a session may or may not have been generated. If it is generated, CDCS will wait until the order has been delivered. Afterwards, it should trigger the /writeSessionLogs endpoint. Upon receiving a successful response, it should call the /deleteProxy endpoint. At this point, the CDCS has completed the task of serving a conversation for a customer and driver for the duration of the delivery.

If no session was generated after /checkService, then at this point, a service has been created for the driver, but no session was generated. Then, the CDCS should trigger /generateProxy to create the session. If it is generated, CDCS will wait until the order has been delivered. Afterwards, it should trigger the /writeSessionLogs endpoint. Upon receiving a successful response, it should call the /deleteProxy endpoint. At this point, the CDCS has completed the task of serving a conversation for a customer and driver for the duration of the delivery.



API Lifecycle

Future Development and Next Steps

Due to the limitations of the Twilio trial account as well has THD order data, there are two areas of improvements that are necessary before the CDCS can serve at scale.

First is the Twilio proxy number management. Because the trial account only allowed usage of one proxy number, we did not implement the concept of grabbing proxy numbers from a pool. However, what we did implement is the functionality to determine if a number is in use or not. Thus, the future team will have to implement a feature such that the CDCS searches through a pool of numbers for a number that is not in use. Additionally, since a number can have multiple conversations, the future team will have to implement another feature to check if a number is reusable in another conversation simultaneously. In other words, if Driver A and customer B are using a number, can Driver C and customer D use that same number (the answer is yes). The groundwork for this has been partially set and should be doable.

Second is the order data. For the purposes of this project, we created a fake database that stored order IDs along with the customer data and the delivery driver data. However, such a repository of information may not actually exist anywhere in THD databases. Therefore, the future team may have to take on another mini project in developing a service that links the driver delivering an order to the customer who placed that order. Then, the CDCS can function as intended. There was little focus on this during the semester, so this a rather larger component that may need its own agenda and team.