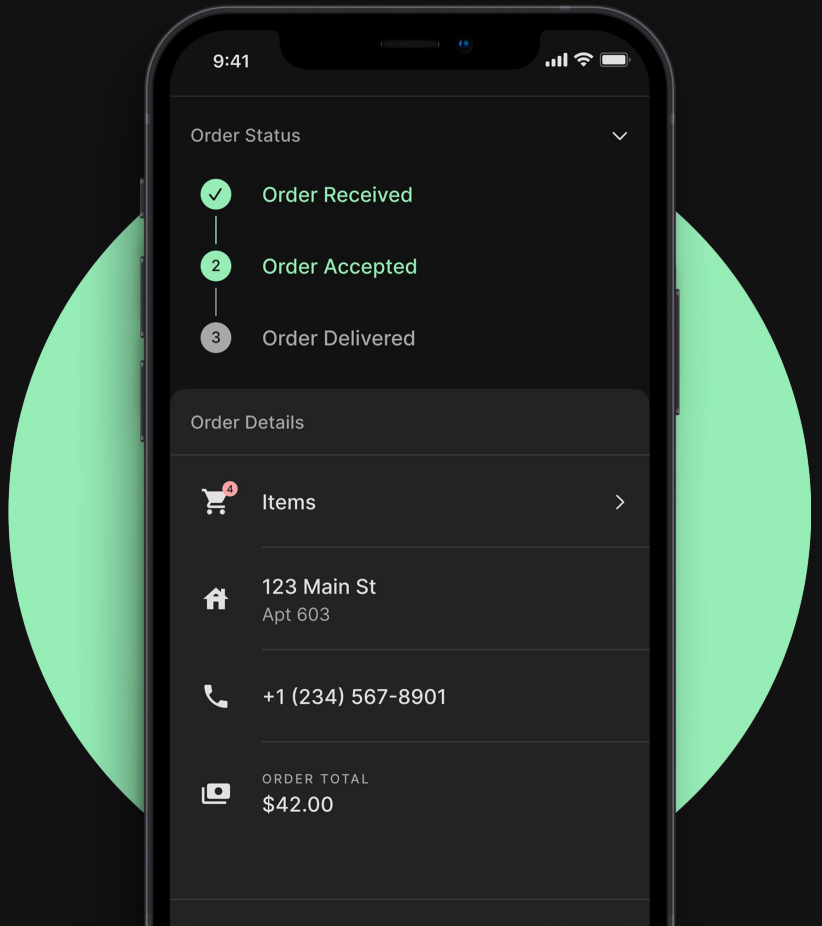
Acolyte

Order Dispatch System

Real-time multi-channel order updates
for an on-demand delivery service



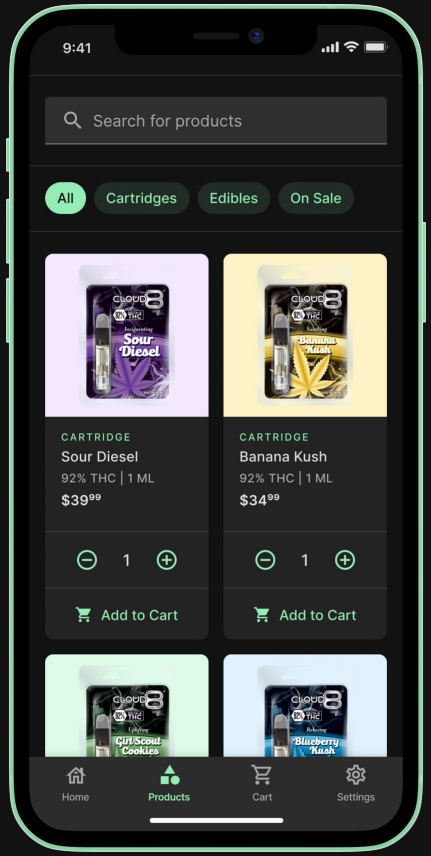
Project Overview

The Product

Acolyte is a Massachusetts-based on-demand cannabis delivery startup (as permitted under the Agriculture Improvement Act of 2018) with a client facing web app. Customers can browse products, purchase items and have them delivered. The fulfillment of orders is performed by independent drivers, using a separate driver web app.

Project Duration

March 2021 to April 2021.



Project Overview

The Problem

Customers and Drivers are experiencing miscommunication in the ordering process.

The Goal

To keep both Customer and Driver updated on the status of each order.

Understanding The User

- Research Summary
- Customer Pain Points
- Driver Pain Points

Research Summary

After interviewing both Customers and Drivers we gained a few insights. Both these user groups have different problems, but they are all centered around the lack of clarity and communication in the current system.

Both Drivers and Customers want updates on each order.

Customer Pain Points

1

Confirmation

The user need a confirmation that the order is being processed after placing it.

2

Receipt

The user needs a proof of purchase: a receipt.

3

Updates

The user need to get updates on the status of their order.

Driver Pain Points

1

New Order

Drivers need to know when a new order has been placed so that they can fulfill it.

2

Complete Order

Drivers need to have a way to signal to both the user and other stakeholders that the order has been fulfilled.

The Solution

- The Goal
- Process Flow
- Technologies

The Goal

1

Status Updates

The user need a confirmation that the order is being processed after placing it.

2

Real-Time

Updates need to be reflected in real-time to keep the user up-to-date.

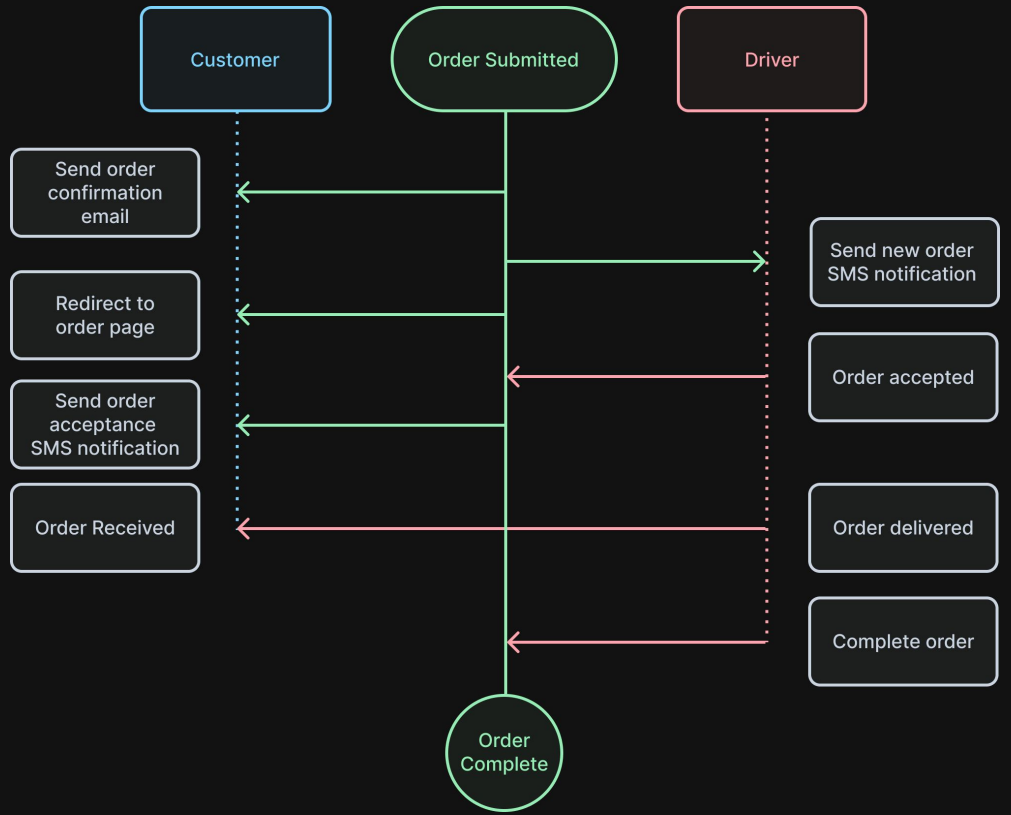
3

Multi-Channel

Multi-channel communication. This means Email, SMS and updates on the app UI.

Process Flow

Diagram visualizing sequence of activities.



Technologies

We chose technologies that would allow us to implement our solution fast and smooth by leveraging specific benefits of each decision.

Firebase Services




Cloud Firestore

1. Cloud Functions to handle the dispatch of each event
2. Firestore to provide real-time updates in application UI

API Services

1. SendGrid for sending order confirmation receipt
2. Vonage to communicate order updates via SMS

Implementation

- Cloud Functions
- Firestore DB
- Email
- SMS

Cloud Functions

We used **Firebase Cloud Function Callables** to make authenticated contextualized requests directly from the client. Doing this instead of allowing the client to directly mutate objects ensures integrity in the ordering process.

We defined 3 functions that would act as the core of the order processing system. To start the process, a customer places an order, calling the `createOrder` callable. This creates an order and notifies the driver of a new order.

“The Cloud Functions for Firebase client SDKs let you call functions directly from a Firebase app.”

Firebase Documentation

Cloud Functions

1

`submitOrder()`

Creates a new Order object in Firestore DB, giving us more control over orders.

Sends email confirmation & SMS notification.

2

`acceptOrder()`

Updates order status in Firestore for real-time update in UI and sends SMS to customer, notifying their drivers name and ETA.

3

`completeOrder()`

Updates order status in Firestore for real-time update in UI. Hands over event to post-sale.

Firestore DB

By using Firebase Firestore database we gained a few advantages right out of the box. Firestore is updated in real-time and synchronized across all connected devices allowing us to provide live updates in the app.

Firestore and Firebase Functions are integrated and works seamlessly together. This made it easy for us to buth present relevant data in UI, but also control the order processing flow from our cloud functions.

“Cloud Firestore uses data synchronization to update data on any connected device.”

Firebase Documentation

Email

Receiving a receipt or proof of purchase was important to the customer. We solved this by using **SendGrid** to send the user an order confirmation email directly after placing the order. This was triggered directly by the submitOrder cloud function.

SendGrid API is simple to use and allows for pre-defined templates to be populated by referencing a template ID and passing it data. This makes it easy to change the email designs in the future.

SMS

A large part of the notification system was achieved by using the **Vonage SMS API** to send text messages to the customers and drivers. They offer many flexible services and a simple API that we called in our cloud functions.

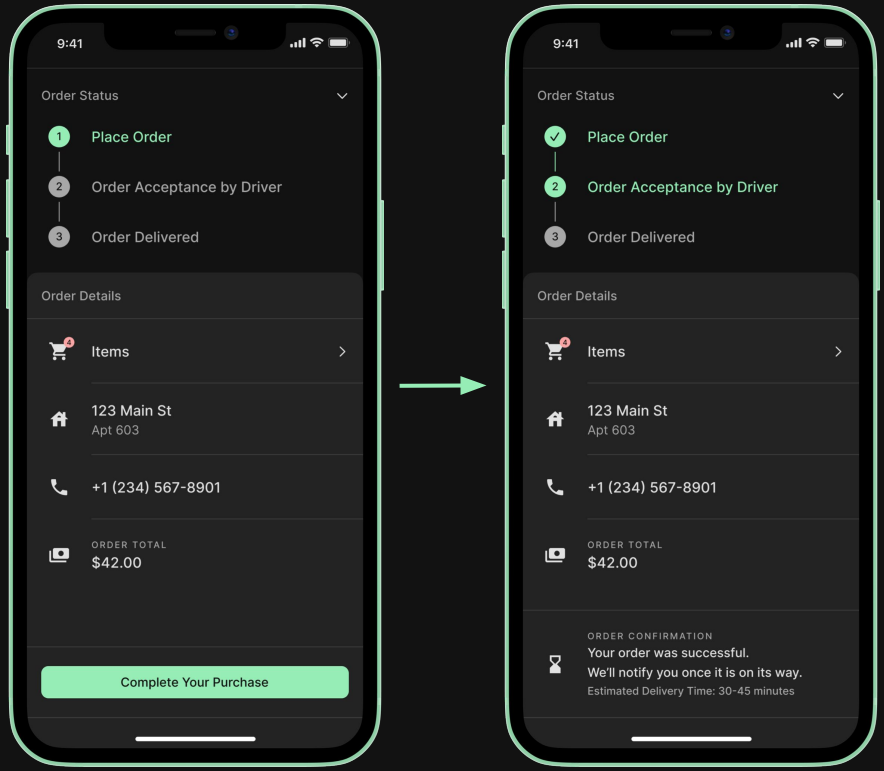
When the submitOrder cloud function was called the system dispatched an SMS to the driver to notify of a new order. When the driver accepted the order with **`acceptOrder`** the customer gets sent an SMS with the drivers name and ETA.

Results

- Demo
- Summary

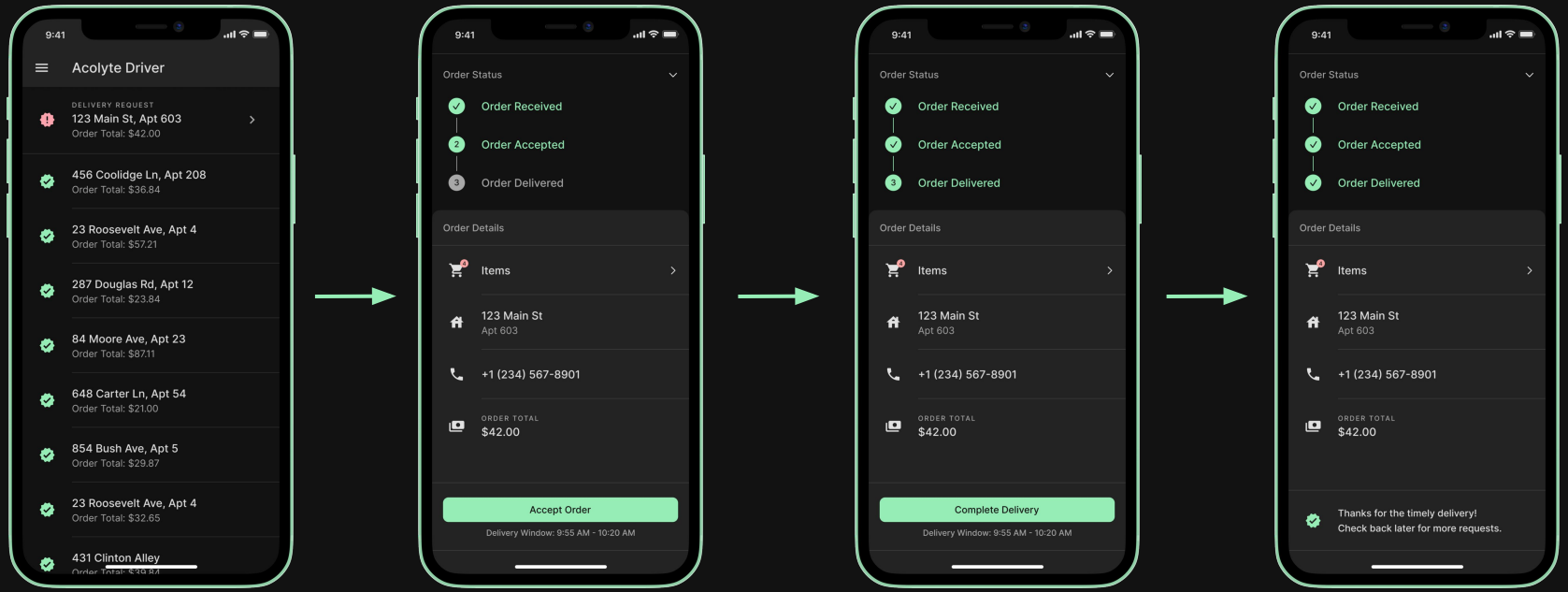
Demo: Submit Order

When a customer completes a new order they're taken to the order page. Here they will see real-time updates in the order stepper.



Demo: Submit Order

When the customer submits an order, the driver app receives a real-time notification showing that order. The driver can then accept it and finally complete it. This is updated in real-time for the driver and customer.



Summary

We intentionally kept the scope of this case study limited to the order dispatch system. There are many other parts we did not mention, such as auth and e-commerce solution, that were foundational for the order dispatch system to work.

We reached our goals of building a solution that keeps the customer and driver updated in real-time. Users now have a clear communication strategy from start to finish when ordering products.

Thank You!