Brian Black

Mychal Calderon

Lexi Christiansen

Final Project write-up

Statement of the Problem

Create a network of flower shop delivery services in an event based distributed system. We allowed for an interface of flower shops and delivery companies work together to create a large network of flower deliveries. To solve this problem, it was important to find out how to have multiple flower shops deliver to multiple homes through multiple delivery services. This modular way of connecting companies with their customers could then be used by many other companies as a delivery API.

API’s

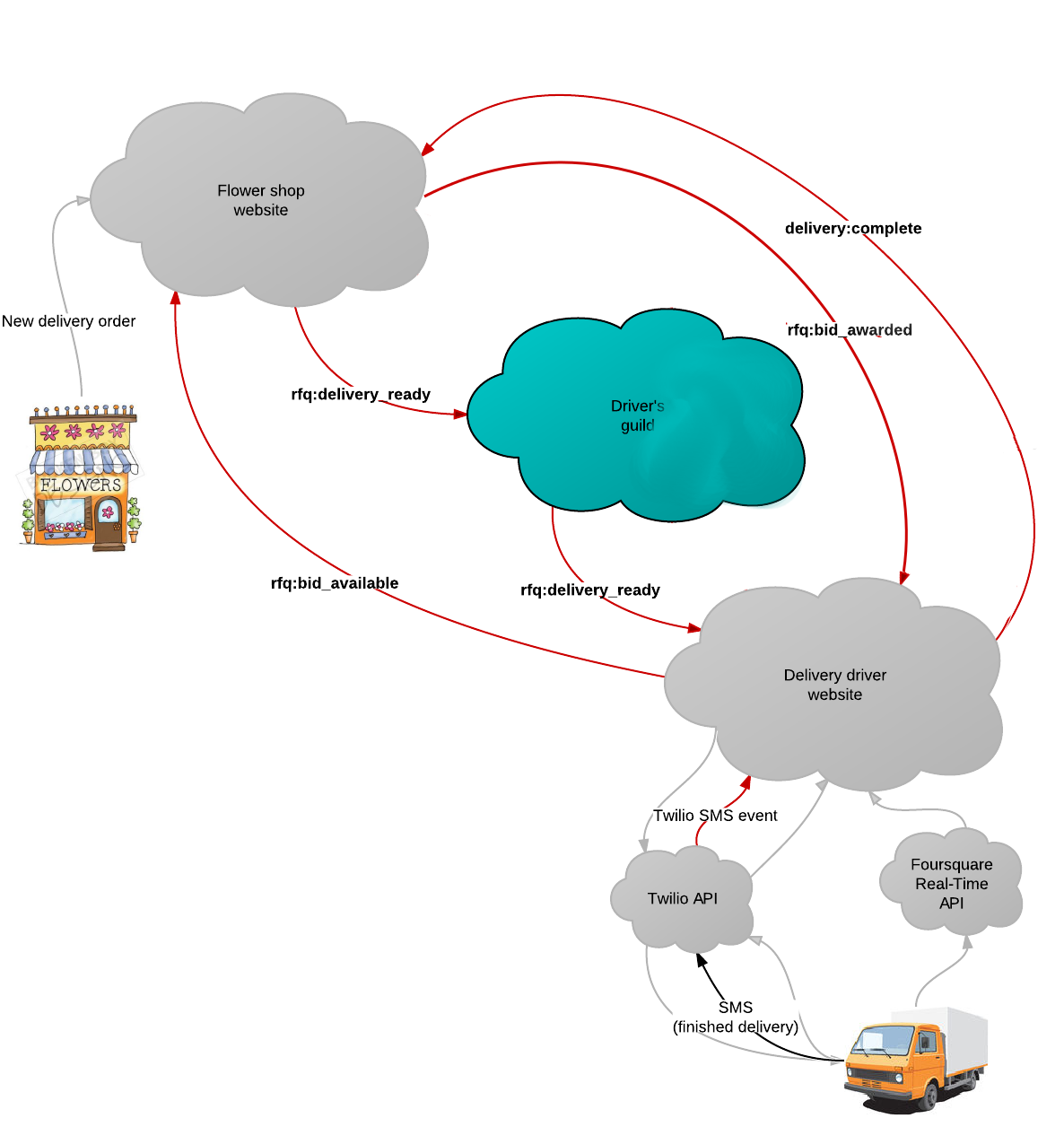
We chose Foursquare and Twilio for quick, real-time updates about delivery locations and when deliveries were completed. This allowed users to find and receive data quickly and easily through a push of data to consumers and companies.

Events

We chose two different types of event domains: rfq (for request for quote) and delivery. The rfq events were for any talk between a flower company and a delivery company. Through these events, we discussed bids for delivery and delivery times. The delivery event domain was for discussing where a delivery was at any given time. We could notify the customer whenever there was an updated status in the delivery route. This included when items were picked up, in transit, or had arrived:

* Delivery:complete – Generated by the driver on a delivery successfully reaching the customer. Consumed by the flower shops.
* rfq:bid\_avaliable – Generated by the delivery companies when they send information to the flower shops that there are drivers ready to work with the driver’s estimated delivery time and rate of service. Consumed by the flower shops.
* Rfq:bid\_awarded – Generated by the flower shops when sending an update to notify a driver that they won the delivery bid. Consumed by a driver.
* Rfq:delivery\_ready – Generated by the flower shop by sending an update to the deliveries waiting for bids that there is a new delivery ready. Consumed by the drivers.
* Text message – Generated by the flower shops and delivery companies when updating delivery status. Consumed by the customer.

Diagram



Method

We went with the micro-framework flask. This allowed for a Python wrapped website with the PyPub Api for using event passed programming with Python and a Sqlite3 database.

We chose Flask as our framework because python allows simple scripts to accomplish large work in a short amount of time. For example, a simple script could pair a flower delivery with a delivery company that works closest to the consumer. This simple setup also allowed for more efficient navigation between web pages and allowed for new features or pages to be quickly added

The PyPub subscriptions allowed different components to subscribe to events in a fashion similar to KRL with listeners and receivers. We made our listeners generic enough that it would be useful for outside users to send requests and made our event calls simple enough that they would be found by either our flower shop sites or our delivery company sites.

We chose Sqlite3 as our database; it was lightweight and good for a small/module system. This worked in our favor when making a way to join multiple shops and delivery companies to better serve the consumer. Our database did not require a large amount of data about the flower shops or the delivery companies. Each database only needed data about where a bid was placed and what company accepted the bid.

Analysis of Events

These events allowed for a non-intrusive way of keeping information flowing through simple push commands to consumers. The passive arrival of event calls allowed for a more streamlined external API calls. Event arrivals were unobtrusive when being sent by other systems.