Design discussion:

1. Backend / Infra / Server side:
    - Regarding we will have one million users, let's say every day for each user query 3 times the data for local food trucks, the average QPS is like: 1,000,0000 * 3 / 86400 = 347, and in peak time like during lunch hours 11:30AM - 2PM on workdays, the QPS could be **tenfold**. Thus, I would use two big RDS instances as well as one instance for backup to handle this, thus in total 3 RDS instances needed to store data of trucks.

    - Backend work role embedded in DB system, we should have some scheduled tasks, who work between our DB and Socrata API from time to time, enabling database to periodically sync with government data using Socrata API, I would say we only need to sync every hour to avoid request rejects, in terms of food trucks info are not supposed to be updated very frequently, as it looks definitely a read heavy system.

    - Given that we already able to sync with data every hour, regarding the scale is big, We could use distributed cache server (**Redis** etc.) which are tend to close to user sides, for example, located on SF downtown CBD, in this way the amounts of queries can be going to cache server, instead of harassing backend DB frequently, also, the latency is expected lower with caching data. On the other hand, we need to ensure the cache is “updated” within acceptable stale time, which requires us to always update cache, when DBs get some rows update, we could definitely use push model to positively update cache during daytime when anything update on DB in a acceptable delay time (like: 30s), and leverage pull model to let caching server to poll the DB during night time (expect much less customer traffic loads) to catch up any new update.

2.Client side
    - Assume most users from mobile, let’s implement it as a simple web app using ReactJs + html5, then enables clients can query either cache or DB using through provided REST API over https.