**About**

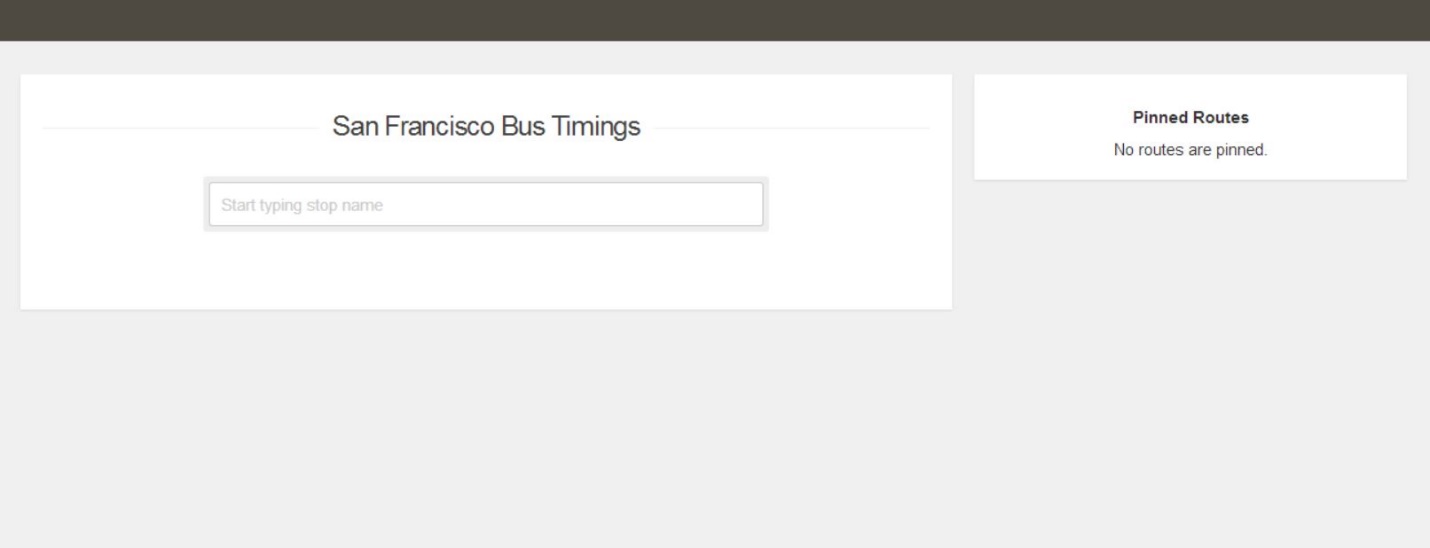
A web application for getting real-time arrival times of San Francisco Municipality bus services. <http://sfbus.azurewebsites.net/>

**Source Code**

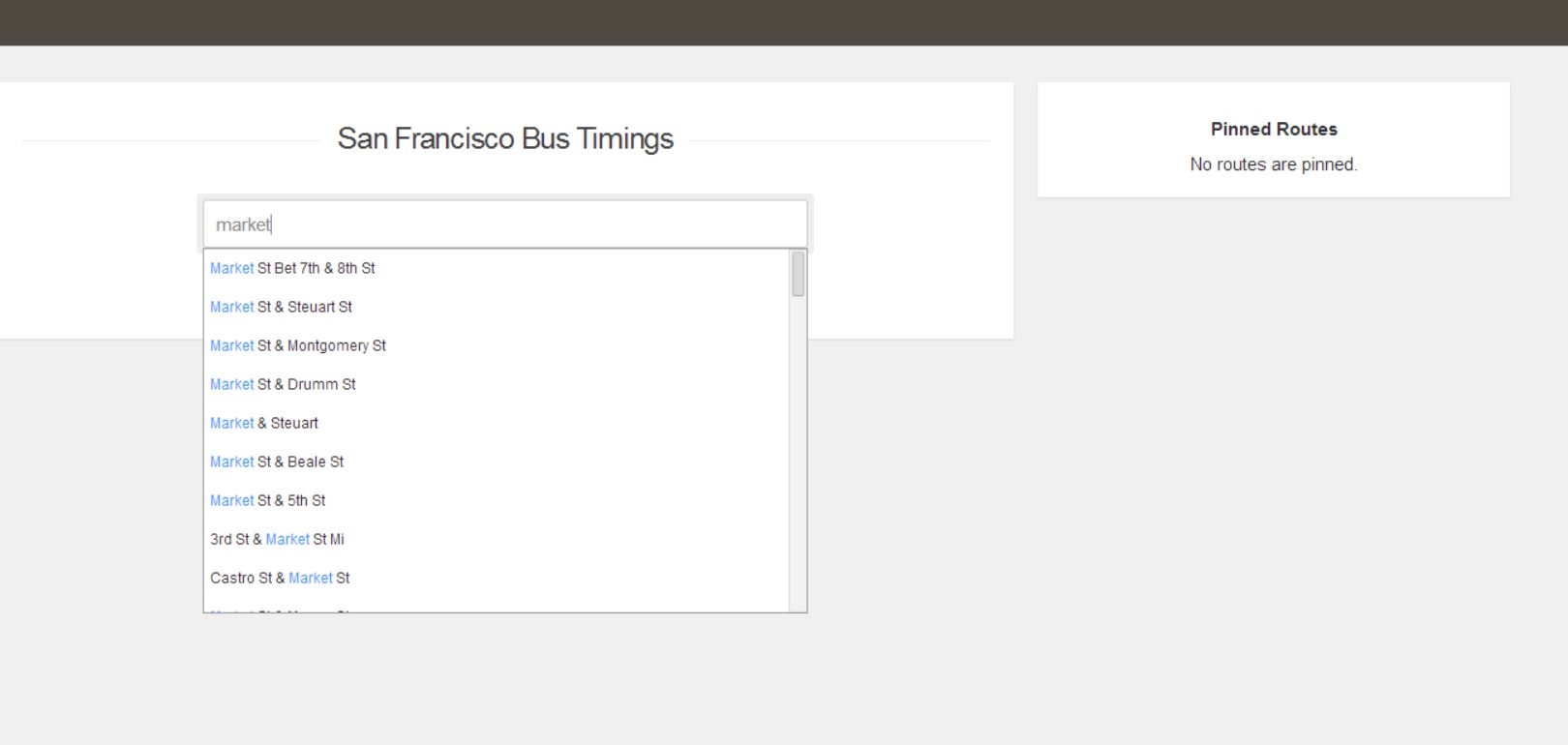
<https://github.com/pranip/sfbus>

**Usage**

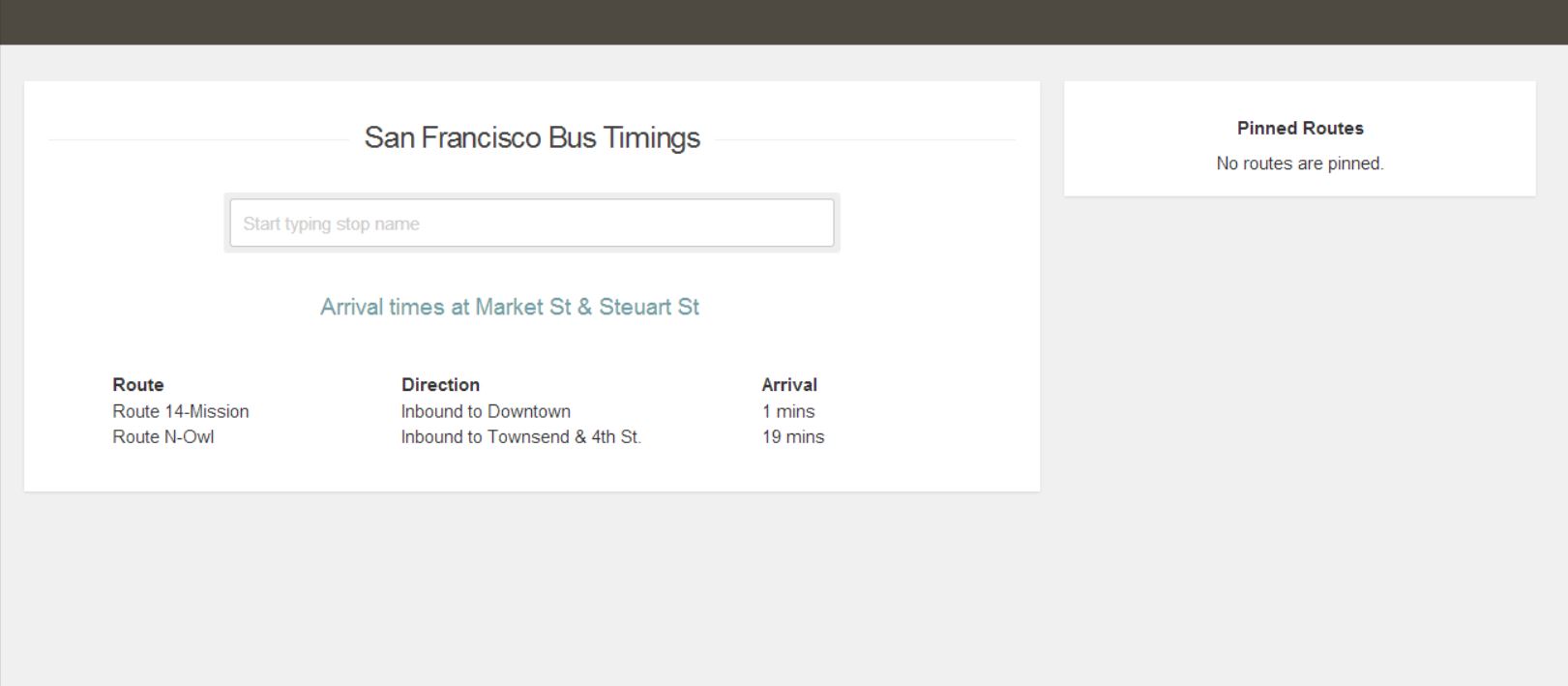
1. Go to the above website.



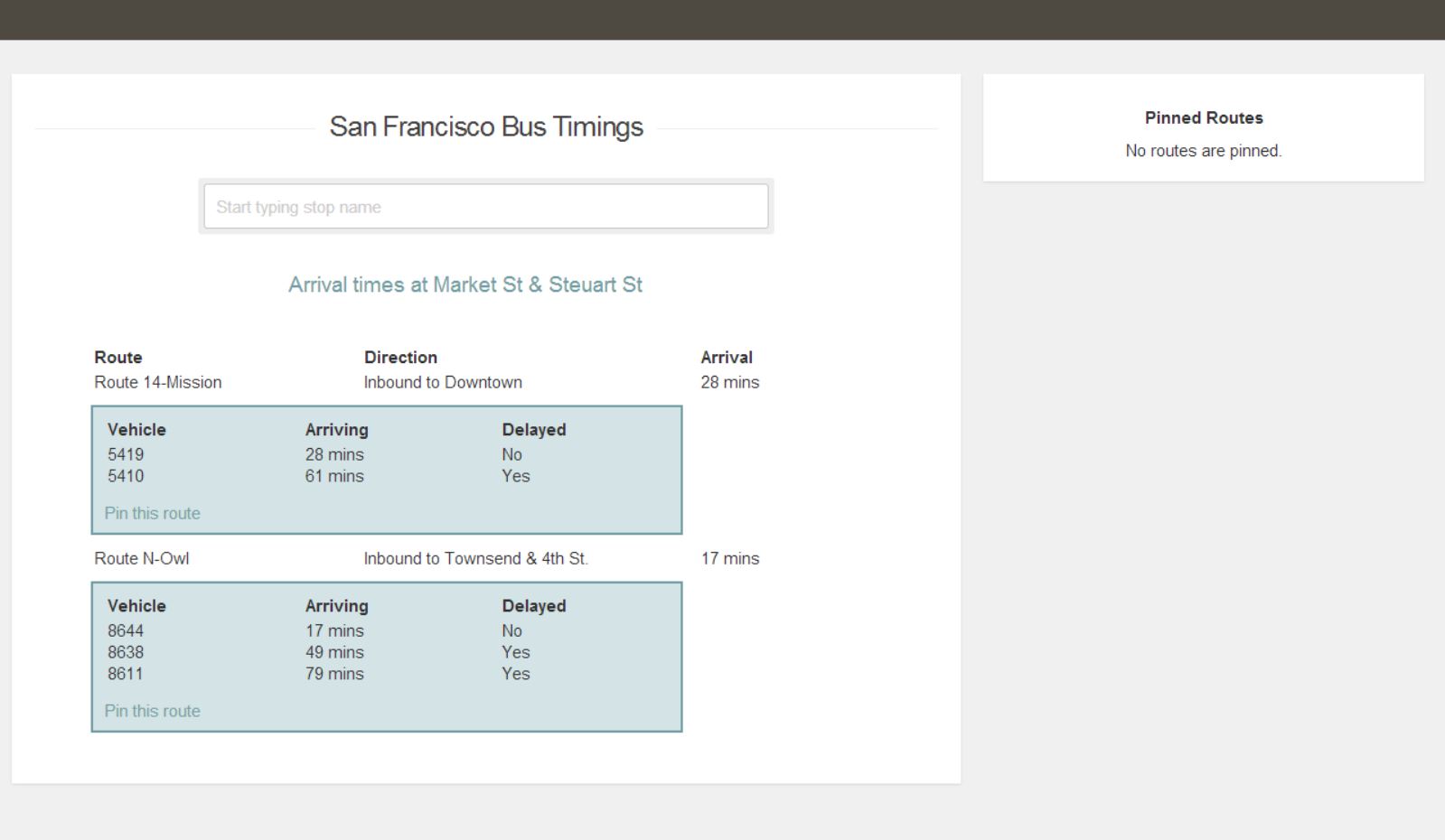
1. Start typing on the search box like street names to find the stop.



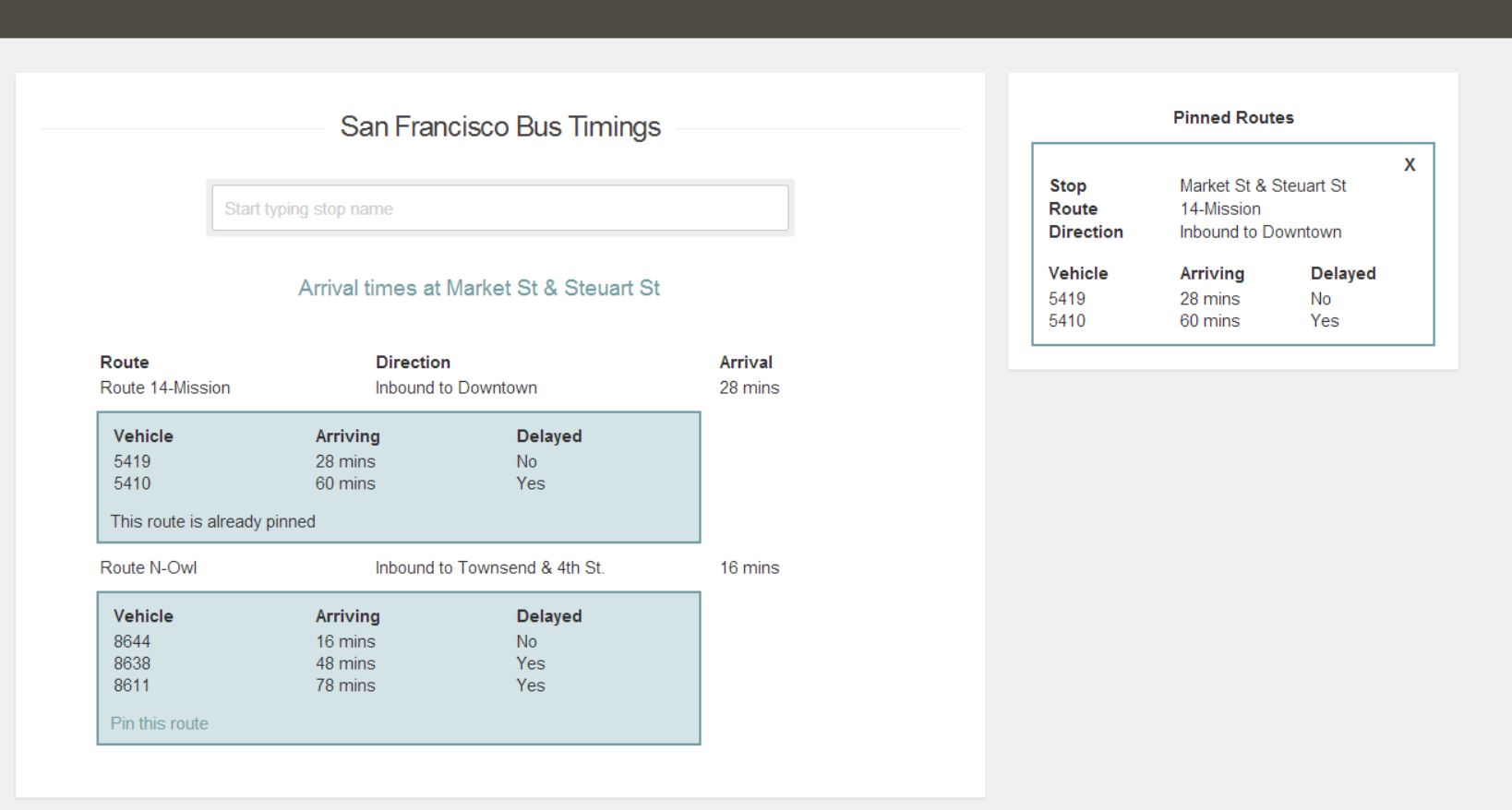
1. Select the stop. It will show the arrival times of all the bus routes for this stop.



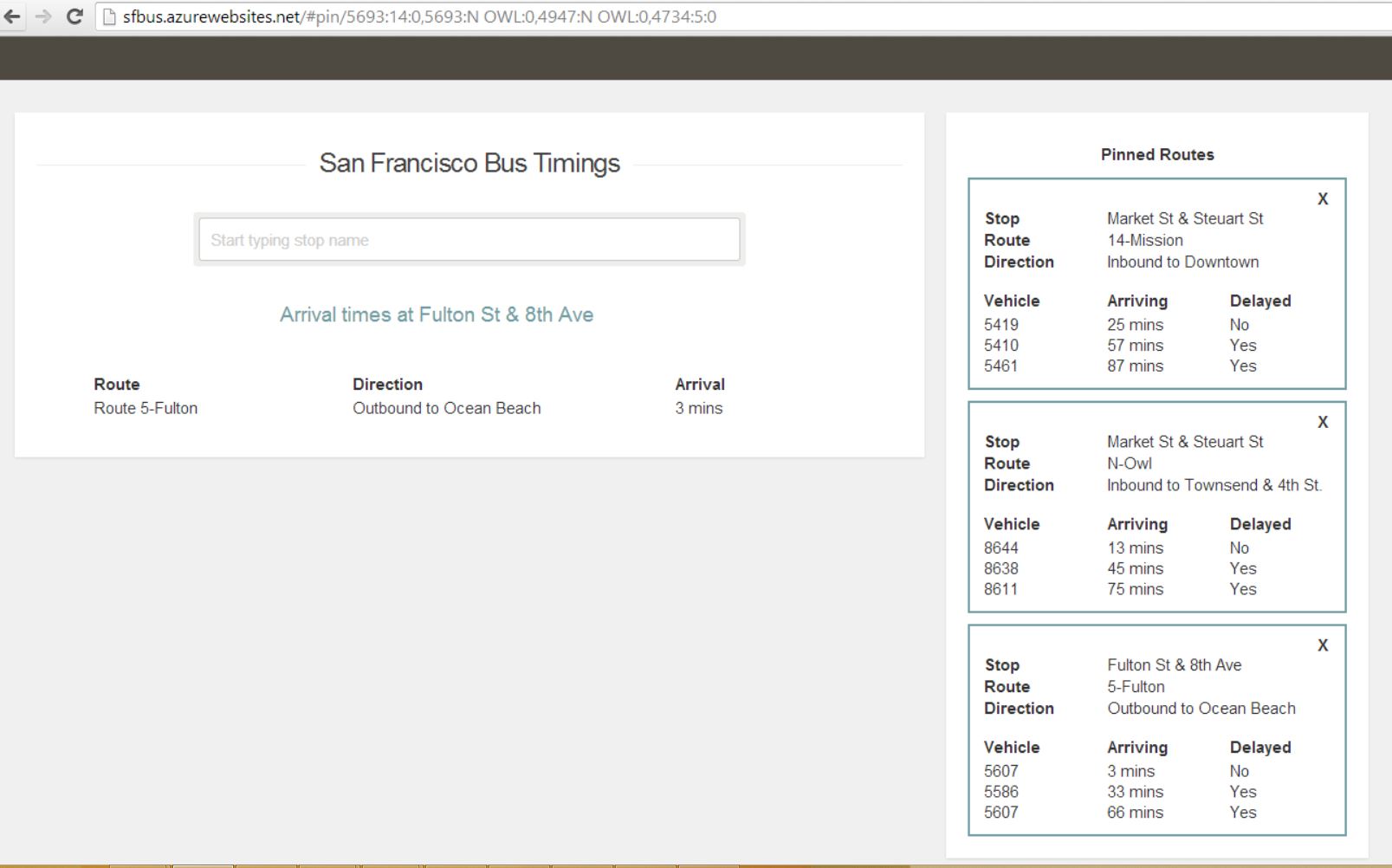
1. Click on each route to find more details.



1. Pin the route to add it in the pin board to the right.



1. Keep pinning different bus routes in the pin board. Or remove it. Watch the URL getting updated without page refresh.



1. Copy the URL and open it in a separate tab. The routes pinned in the older tab shows up in the new tab too.

**Architecture**

**Front End:-**

1. JS/CSS/HTML/jQuery.
2. [Backbone](http://documentcloud.github.io/backbone/) for UI framework
3. [Underscore](http://underscorejs.org/) for UI template system
4. [JQuery plugin auto-complete](https://github.com/devbridge/jQuery-Autocomplete) for auto-suggestion of stop names.
5. [Socket.IO](http://socket.io/) for real time data transport using web sockets or polling based on the browser support.

**Back End:-**

1. Node.js
2. NPM modules installed

"express": "3.4.4",

"jade": "\*",

"stylus": "\*",

"socket.io": "~0.9.16",

"xml2js": "~0.4.1"

**Why used above?**

Mostly because I wanted to learn about them. I wanted to use this as an opportunity to get hands on experience on the above technologies/frameworks.

**Web Services Used:-**

<http://www.nextbus.com/xmlFeedDocs/NextBusXMLFeed.pdf> for fetching the route, stop, vehicle and arrival time information.

**Hosting**

Hosted in one instance of Microsoft Azure standard Webrole

**Design constraints**

**Next Bus API:**

1. Provides real-time bus predictions for many bus services across US.
2. It’s data is organized in the following manner –
3. Agency: There are many agencies. We are interested in only one agency- i.e. sf-muni

API: <http://webservices.nextbus.com/service/publicXMLFeed?command=agencyList>

1. Routes: Each agency has multiple routes

API: <http://webservices.nextbus.com/service/publicXMLFeed?command=routeList&a=sf-muni>

1. Stops: For each stop, arrival times of bus routes passing by this stop can be fetched

API: <http://webservices.nextbus.com/service/publicXMLFeed?command=predictions&a=sf-muni&r=F&s=4504>

Bulk API <http://webservices.nextbus.com/service/publicXMLFeed?command=predictionsForMultiStops&a=sf-muni&stops=N|6997&stops=N|3909>

**Next Bus API restrictions**

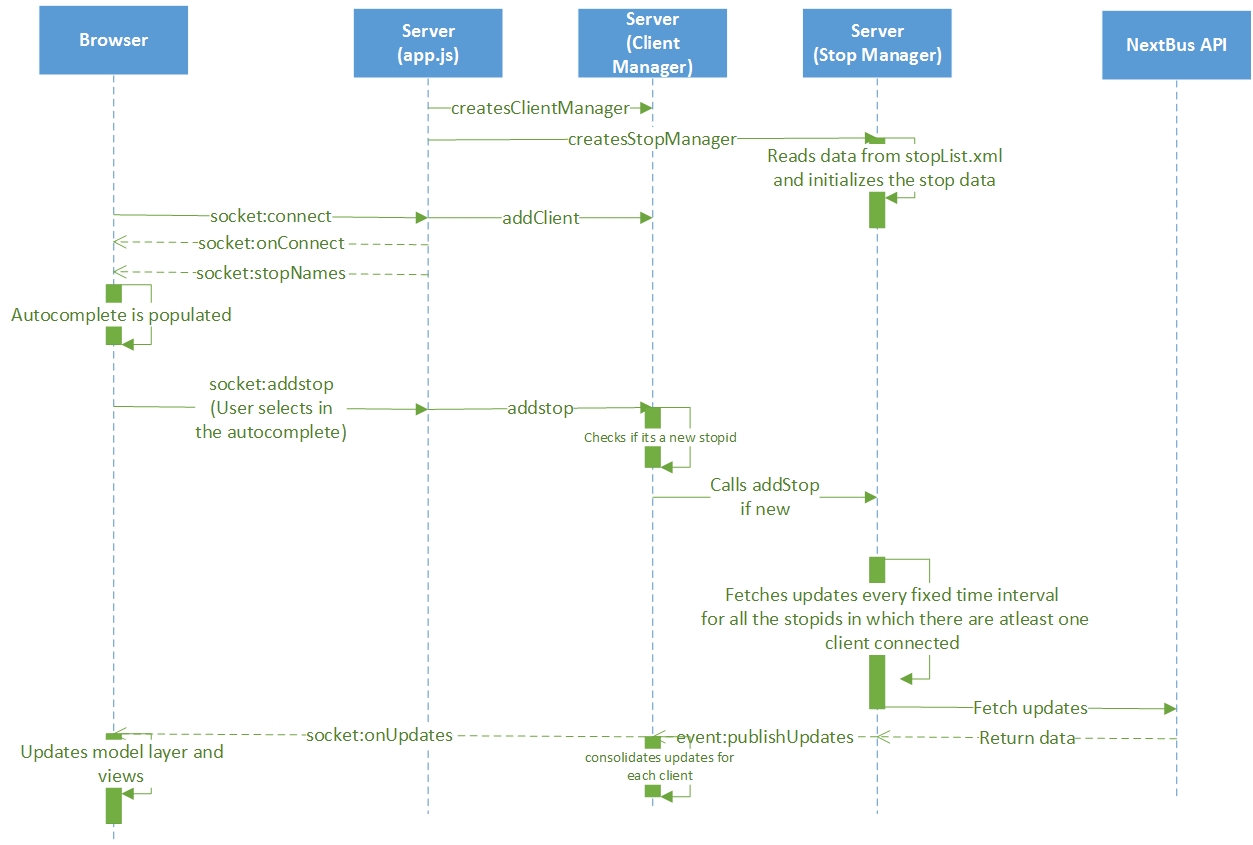
1. The Next Bus API has quota limit on number of calls and data that can fetched.
2. Every user directly calling to the service from the browser will bring extra traffic to the API.
3. Using batch call is more optimal than multiple single calls to the API. This helps in getting the maximum benefit within the quota restrictions.
4. Next Bus API is top down approach where the user needs to know the bus service name, then route and then finally shown bus stops for a given route (Try it here <http://www.nextbus.com/> ). It’s a three step process.

Whereas, we need our application to be one step where user only searches for the stop and we show all possible routes for this stop.

**Goals of Back-end service**

1. Act as a broker between the client (browser) and the API so that calls to the API can be batched, controlled according to the quota limits.
2. Funnel back the updates to the client based on the updates received from the API so that the client don’t not need to keep polling the API. Using web sockets, clients are informed as and when updates need to be sent.
3. Some level of pre-processing / data transformation is needed to be able to query able by only searching stops and showing all the arrival times of all bus routes for a given stop.
4. Through pre-processing, generated this [xml file](https://github.com/pranip/sfbus/blob/master/Files/stopList.xml) . This is a list of all stops in the SF Muni agency and storing route ids for all buses passing through that stop.

**Sequence Diagram**

****

ClientManager**:** Manages clients. Each client in one browser session (one socket connection). Every client subscribes for updates for given set of stops. This class maintains the list of stops for every client and informs StopManager of new stop. When a client is disconnected, it removes the information of the client and also informs StopManager to remove the stops. It filters the updates received from StopManager, and sends it to the client based on whichever stops the client has subscribed.

StopManager: This class manages list of all stops for which there is at least one client that has subscribed for this stop. If there are no client interested for a stop, that stop is removed. For a given set of stops at any given time, it periodically fetches the updates by making bulk call to the nextbus service. This class ensures the calls made to the external service (nextbus) is optimized by making sure:-

1. Doesn't exceed the throttling limits *(no limit set presently)*

2. Based on throttling limit and current load, adjusts the polling interval. *(Not implemented the logic)*

3. Makes bulk batch call to optimize on the throttling limits.

After every fetch call it publishes updates by firing event publishUpdates. The ClientManager, which binds to this event, filters, organizes and pushes the data for each client.

**Goals of Front-end**

1. Single Page Application (Use Backbone for MV\* framework)
2. Real-time data transfer using web sockets whenever possible (through socket.io)