- User enters destination
- map pops up and displays:
    - current location
    - destination
    - clusters/markers for the closest 200 parking meters.
- user clicks a marker or zooms/moves/etc then clicks a marker (for a meter)
- popup displays meter info
- driving route is drawn from current location to the meter in green
- walking route is drawn from the meter to the destination
- user can click “update” to modify a meter’s values

Objects

RequestMeters:
[
    {
        key: unique key
        lat: latitude (float)
        lon: longitude (float)
        time_limit: total allowed minutes (integer)
        time_per_quarter: minutes of parking per quarter (integer)
        enforcement_start: hour enforcement begins in 24h format (integer)
        enforcement_end: hour enforcement ends in 24h format (integer)
        congestion: value 1-10 estimating how hard it is to park (integer)
    }
]

Update:
{
        key: unique key
        time_limit: total allowed minutes (integer)
        time_per_quarter: minutes of parking per quarter (integer)
        enforcement_start: hour enforcement begins in 24h format (integer)
        enforcement_end: hour enforcement ends in 24h format (integer)
        congestion: value 1-10 estimating how hard it is to park (integer)
}

URLs

/
GET
Returns the “main.html” template

/route
GET -> destination: location string denoting destination to travel to
Returns the “route.html” template

/update
POST -> key: unique key
time_limit: total allowed minutes (integer)
time_per_quarter: minutes of parking per quarter (integer)
enforcement_start: hour enforcement begins in 24h format (integer)
enforcement_end: hour enforcement ends in 24h format (integer)
congestion: value 1-10 estimating how hard it is to park (integer)
Updates the meter data in ndb to reflect the arguments sent with “/update”

/setup
GET -> command: command string
Runs setup functionality based on command string

/request
GET -> new_request: location string denoting approximately where to park
Returns a list of 200 meters closest to the location denoted by “request”

NDB Models

For our project, we used geomodel, a library which allows for geospatial queries. It leverages the ndb.GeoPt().

ParkingMeter
location: ndb.GeoPt(): this is abstracted away by geomodel and represents the latitude and longitude of the meter
time_limit: ndb.IntegerProperty(): total allowed minutes
time_per_quarter: ndb.IntegerProperty(): minutes of parking per quarter
enforcement_start: ndb.IntegerProperty(): hour enforcement begins in 24h format

enforcement_end: ndb.IntegerProperty(): hour enforcement ends in 24h format
congestion: ndb.IntegerProperty(): value 1-10 estimating how hard it is to park