My app URL is http://52.38.127.124:3008/ .That's the main page of the web interface of the application.

The REST API is shown at http://52.38.127.124:3008/api . It has simple instructions for performing basic operations and checking that references to elements are deleted and whatnot.

And the complete list of tests is at http://52.38.127.3008/test . It's a list of cURL calls you can use if you have cURL installed, and you should get the expected results on all 21 tests.

I made a RESTful API that could potentially be used for the app Airbnb, which is an online marketplace that allows people to advertise, find, and rent vacation homes. But unlike Airbnb my app allows people to comment anonymously on listings (people on both ends are usually motivated to give 100% positive reviews to reduce their chances of being rejected in the future).

I mostly stuck to my plan, using Node.js with Express and MongoDb. However, there were two ways that I deviated: I added the ability to post and remove comments from listings, and I used my Amazon Web Services EC2 server instead of google-app engine.

The reason that I added comments was because with my planned schema, the relationship was supposed to be with "location." But that doesn't really show a relationship, since every listing will only have one location, and you aren't going to independently delete a location. There wasn't much of a point of having it as a separate schema. So I decided to add a comments schema, which can be individually added to and removed from listings, and which are also removed from the database when their parent listing is. In the end my schema looked like this:

```
var locationSchema = db.Schema({
  city: String,
  state: String,
  _id:false
});
var commentSchema = db.Schema({
  words: String,
  listing_name: String,
  listing_id: ObjectId
});
var listingSchema = db.Schema({
  location: locationSchema,
  comments: [commentSchema],
  name: {type: String, unique: true, required: true},
  noGuests: {type: Number, unique: false},
  price: {type: Number}
});
```

The reason that I used my AWS server instead of Google app engine is that, for whatever reason, my Google app engine apps were not able to receive DELETE and PUT requests. I was using the flexible environment, which is still in beta, with nodejs, and I found that the same DELETE and PUT requests that worked fine on localhost would not work on my app engine deploy. Luckily my EC2 server worked fine.

The URL structure that I use to access my resources is very simple. All of these do what you would expect them to:
//routes that use get HTTP request:
/getAllListings, getAllComments, getListingsByLocation
//routes that use post HTTP request:
/postListing, postComment
//routes that use put HTTP request:
/putListing
//routes that use delete HTTP request:
/deleteListing, deleteComment

I tried to make my API RESTful. The REST constrains that my API meets are:

-The client and server are separated. The client cares about the UI, and the server about the data.

-It's stateless. The client has no responsibility towards keeping track of the data, though you can optionally use a web browser to access certain routes that allow you to manipulate data, all operations on data are done solely on the server.

-It is layered. There are a lot of routes involved and the servers involved are both my AWS server and the wherever mLab.com keeps my data. But this is opaque to the client.

Some constraints that it does not meet:

-Resources are not clearly marked as either cacheable or non-cacheable.

-Instead of sending the resources themselves, it sends `res.json(resource)` of the resources. I didn't realize until re-watching the RESTful API lecture that I think a RESTful API is supposed not supposed to return its resources in JSON format, but instead make it so that it can be accessed by XML or whatever the client specifies. So if I were to do it differently, I definitely would have sent the results themselves so that my /get methods could retrieve more formats than JSON.

Another thing that I would do differently is that I would make my code a lot neater. My listing.js file is very long, and the routes in there are named confusingly.

I also would not have made "location" into a separate schema at all, since there is no point in that, and instead would the just used normal "city" and "state" fields for listing objects.

Lastly, I would have made it so that a listing or comment can only be deleted with the same API key that was used to post it, instead of giving all API keys universal access. I don't remember quite what I had planned when I started with API keys but the idea was to prevent people from modifying or removing each other's data.