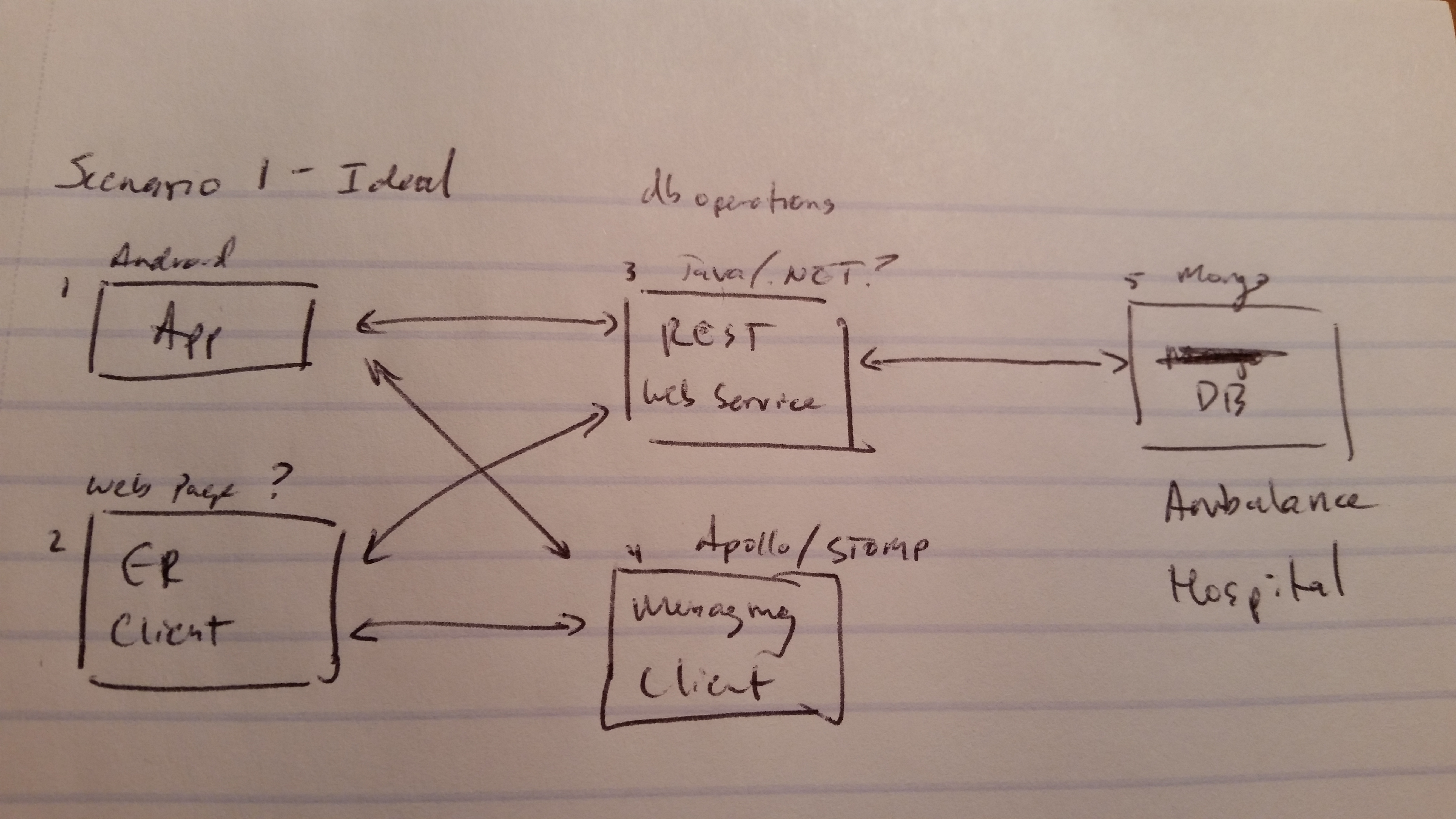
**Ambulance Rerouting Architecture and Design**

Here’s a copy of the system architecture I sketched out earlier:



Let me walk through the basics at each level and summarize what we talked about last night:

**1. Ambulance App**

We decided this would be a web app and John would start working on this layer. This app should do the following things:

* Create an ID and record its ID and initial location in the database.
* On patient pickup, allow the user to input the following criteria:
  + Age (Simplify this to present options of {child|teen|adult} since we don’t want the user to have to actually type numbers here that they may not know initially to begin transport
  + Gender
  + Condition. This will be one of four criteria {level1|level2|level3|basicER}
* Provide this information to the Rest Service and receive a response with the top 3 locations. Data format will be discussed later in the REST portion.
* The user selects the desired location, which triggers a Database Update and a Notification message sent to selected hospital to notify of inbound.
* Plot directions for the ambulance to follow visually. Update location on map at regular intervals (15 seconds) and in database. Display a running ETA for arrival.

Again, John is tackling this topic, and will be using .NET. Also note that we ideally want to send all Database operations through the REST layer, so we will need CRUD support for our queries.

**2. ER Client**

We decided this would also be a web app and Sai Teja would start working on this layer. This app should do the following things:

* Via some sort of login setup, choose which hospital we are representing.
* Read the ER capacity and Trauma capacity, if any, through the DB/REST Service.
* We need to decide if we want to mock up an initial state for the rooms, or have that just be part of our persistent data set. I’ll indicate when I get to the Database portion what kind of options we might have.
* Present a visual layout (probably some sort of grid?) of all available rooms with the Trauma rooms (if present) clearly separate from the standard ER rooms.
* Each room should be presented as in one of three states: {open|occupied|cleanup}
* User should be able to toggle the room between states in order: Open -> Occupied -> Cleanup -> Open…
* The UI should also show a list of all inbound ambulances with the demographics of the patient clearly visible.  
  This means it should show all the elements provided above (Age, Gender, Condition). In addition, it should show the ETA for the Ambulance.
* The User should have the ability (by button press with action confirmation) to order an ambulance to reroute. This should dispatch a message to the ambulance, resetting its assignment.

We decided this will be a web page with most of the relevant data being provided by our REST service. Presumably this will be in HTML5 supported by JavaScript and jQuery/Bootstrap.

**3. REST Service**

The REST Service must be able to do 2 things:

* Support CRUD operations for all Database entities.
* Implement the ‘nearest hospital’ algorithm.

To do this, we need to support 3 service handles (we don’t need to stick with these names, but the idea is the same):

**/ambulance**

**/hospital**

**/closestpath**

**/ambulance** will need to support the following REST operations:  
**GET: /ambulance**

PARAMS: id=<ambulance id value>

hospital=<hospital id value>

GET /ambulance?id=42

Would return all the relevant data for ambulance number 42

GET /ambulance?hospital=21

Would return a list of all ambulance data assigned to the provided hospital id.

**PUT: /ambulance**

BODY: An Ambulance Object (in JSON form) to be updated into the database.

**POST: /ambulance**

BODY: An Ambulance Object (in JSON form) to be inserted into the database.

**DELETE: /ambulance**

BODY/PARAM: An Ambulance Object (or we could simplify this to just be the ambulance ID) (in JSON form) to be deleted from the database.

This would be done when the ambulance has ‘delivered’ its patient. You can decide how you want to handle the format of this request.

**/hospital** will need to support the following REST operations:  
**GET: /hospital**

PARAMS: id=<hospital id value>

GET /hospital?id=21

Would return all the relevant data for hospital 21.

GET /hospital?id=0

Would return a list of all hospitals. Not sure if we’ll need this, but we should probably leave this open as a possibility.

**PUT: /hospital**

BODY: A Hospital Object (in JSON form) to be updated into the database.

**POST: /hospital**

BODY: A Hospital Object (in JSON form) to be inserted into the database.

**DELETE: Unsupported**

We definitely do not want to be deleting our hospital set, so this should either not exist or if implemented throw some kind of UnsupportedOperationException.

**/closestpath** will need to support the following operation:

**GET: /closestpath**

PARAMS: amb\_lat=<latitude coordinate of ambulance>

amb\_lon=<longitude coordinate of ambulance>

age={child|teen|adult}

condition={level1|level2|level3|basicER}

exclude=<hospital id(s) to exclude> *(Optional)*

GET /closestpath?amb\_lat=-32.234&amb\_lon=45.3487&age=teen&condition=level3

This example should be what is needed to give the computation algorithm everything it needs to perform its calculation.

The return format in JSON should be something like this:

[{“rank”:1,“hospitalID”:21,”name”:”Liberty Hospital”,”lat”:-32.453345,”lon”:45.225235,”eta”:”10:37”}, {“rank”:2,“hospitalID”:17,”name”:”North Kansas City Hospital”,”lat”:-32.453345,”lon”:45.225235,”eta”:”12:37”},

{“rank”:3,“hospitalID”:13,”name”:”Saint Luke’s Northland”,”lat”:-32.453345,”lon”:45.225235,”eta”:”13:41”}]

We want the algorithm to return the top 3 results ranked by ETA. Ashok will be working on building out the REST Service using .NET.

**4. Messaging Service**

We need some kind of messaging queue system that can send real-time notifications either to the hospital that a new ambulance is inbound or that a hospital has requested an ambulance divert to another location.

There are then two basic messages:  
**Ambulance**: I have chosen to proceed to Hospital X.

**Hospital**: I need to request Ambulance Y to reroute.

There could be some additional messaging as we progress with the project such as:  
**Hospital**: I acknowledge that Ambulance Y is inbound.

To do this, we want to implement the Publish/Subscribe Design Pattern, where a central messaging queue system supports subscriptions from Ambulances and Hospitals and routes messages to the intended targets.

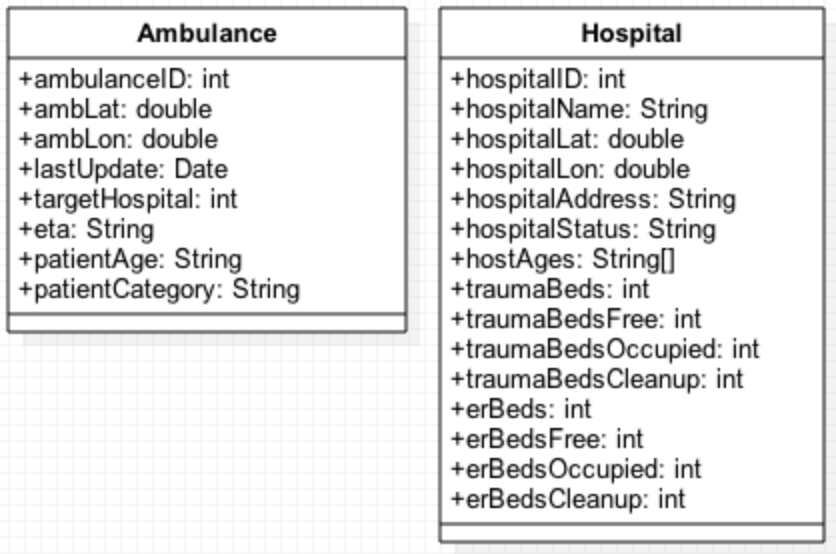
There are many open source options available. Some of the most promising are Apache Apollo, RabbitMQ, and Google Messaging. For the first step of this project, we need to figure out how to create a messaging queue, and create two simple web pages that use JavaScript/AJAX to subscribe to the messaging queue, can send messages back and forth, and can reflect changes in the UI based on the message received.

Alex will be working on this piece. He will need to decide on a queue system and implement the proof of concept web sites so that this functionality can then be incorporated into the Ambulance and ER sites.

**5. Database**

We will be using MongoDB, a NoSQL database engine, to host our rather small data set. Our database needs to represent two entities: Ambulance and Hospital.

Presented below are the schema (with some optional elements discussion) for our two entities:



I feel like most of the fields for Ambulance are self-explanatory. The lastUpdate column should act like a Timestamp to indicate the last date the Ambulance posted its location or data.

Many of the Hospital fields are also self-explanatory, but I will explain those that might not be.

* hospitalStatus will be where we indicate whether this is a {level1|level2|level3|basicER} service provider.
* hospitalAddress will be the text string of the address, primarily for display purposes, but which can also be submitted via Google Maps api, though Lat and Lon values should be preferred.
* hostAges is a list of which of the three age groups this hospital can serve. Most should possess all three – [“child”,”teen”,”adult], though some like Children’s Mercy may only possess [“child”,”teen”]. The MongoDB $in operation can be used to make sure we eliminate hospitals that do not meet our criteria.
* traumaBeds\* and erBeds\*. The first traumaBeds and erBeds fields are required, as they set capacity values for each hospital. Adding in the \*Free, \*Occupied, and \*Cleanup fields will allow us to persist a state for each hospital, and should probably factor into our nearest hospital algorithm. For instance, a Patient requiring level1 care should not be assigned to a hospital where traumaBedsFree = 0.

Adam is working on the MongoDB data set creation.

This should be enough for us to be off and running. I hope this gives everyone a clear idea of the how the pieces fit and what everyone should be working on. The schema models should be directly translatable into Class objects, so we can be consistent across the various layers.