Server Interactions

Robodex

This document uses the Application Flow diagram as a reference. The numbered screens used in this document directly correspond to the numbered screens in the diagram. This is written from the perspective of the app.

Implicit with each "Send:" is a request\_type field that is the name of the request being sent from the app. Also, the user's session token, after they have logged in. Implicit with each "Receive:" is an error\_code and error\_message field that contains any additional information about the response from the server. Error codes and messages are defined in the Error Codes spreadsheet.

A question that keeps coming up as I write this is how much data should the server return in a request. For example, when returning roles, the role\_id and role name would be sufficient to list the roles. However, the roles table is rather small and returning all of the data so it is available when the user clicks a role would make more sense. Another example is when returning a list of people with a given specialty. Including all of each person's information here would probably be slower than doing it in two separate queries (thus getting all of the information on only one person). It's a trade off that is present in the majority of the requests; it deserves some thought.

I just realized there is no mechanism for adding or deleting roles, only editing them. I think that's okay for now though.

**\*** Note for returning arrays: the size of the arrays could grow very large. When returning an array, it should be limited to -- lets say 50 results (will need further testing to see how many we can return before slowing down the programs). When the user scrolls near the end of the list, a new request will be sent for the next 50 or so results.

**Request 1 - Login**

**Send:**

Username/email

password

**Receive:**

session token

**When/Where:**

Screen 1a - When the Login button is clicked

Screen 1b - When the correct PIN is entered

**Request 2 - Get Specialties \***

**Send:**

result to start at (default 1, server determines limit)

**Receive:**

Array of specialties (specialty\_id, specialty)

**When/Where:**

Screen 3 - on create

**Request 3 - Get Coordinates**

**Send:**

min latitude

max latitude

min longitude

max longitude

**Receive:**

organization locations (location\_id, location info, organization name)

people (person\_id, person info, (should this include specialties and organizations?)

members (from check-ins (lat/lon in member table, if the check-in occurred within X amount of time (a day or two, less?), corresponding person info)

**When/Where:**

Screen 4 - on create

**Request 4 - Get Organizations \***

**Send:**

result to start at (default 1, server determines limit)

**Receive:**

Array of Organizations (organization\_id, organization)

**When/Where:**

Screen 5 - on create

**Request 5 - Get Links \***

**Send:**

result to start at (default 1, server determines limit)

**Receive:**

Array of Links (link\_id, link, title)

**When/Where:**

Screen 6 - on create

**Request 6 - Get Pending Removals\***

**Send:**

result to start at (default 1, server determines limit)

**Receive:**

Array of Pending Removals

**When/Where:**

Screen 15 - on create

**Request 7 - Get Roles**

**Send:**

-

**Receive:**

Roles info

**When/Where:**

Screen 17 - on create

**Request 8 - Get People by Specialty\***

**Send:**

specialty\_id

result to start at (default 1, server determines limit)

**Receive:**

People that have the given specialty

**When/Where:**

Screen 8 - on create

**Request 9 - Get Organization Locations \***

**Send:**

organization\_id

user location (coordinates)

result to start at (default 1, server determines limit)

**Receive:**

Locations of the given organization

**When/Where:**

Screen 10 - on create

**Note:**

Listing them according to distance would probably make sense, with one exception. Primary locations (Headquarters) might make sense to be first. If there are lots of locations (>50 perhaps), the server should determine which are closest? I'm trying to think of a way to avoid getting 50 locations, and none of them are the closest. Using a lat/lon box (like in the get coordinates request) wouldn't work if there were only a few total locations. In this situation, the user would get a list of just a few locations that were close, but they should just get the whole list - ignoring distance. Consider this when implementing.

**Request 10 - Get Person Detail**

**Send:**

person\_id

**Receive:**

combined person and member information, appropriate for the user requesting it :)

**When/Where:**

Screen 11 - on create

**Request 11 - Get (Organization) Location Detail**

**Send:**

organization location\_id

**Receive:**

the given location's information, appropriate for the user requesting it

**When/Where:**

Screen 12 - on create

**Request 12 - Get Approved Removals \***

**Send:**

order (default newest first)

result to start at (default 1, server determines limit)

**Receive:**

Items that have been approved for removal

**When/Where:**

Screen 16 - on create

**Request 13 - Check In**

**Send:**

member\_id

user's coordinates

**Receive:**

success or fail

**When/Where:**

Default Menu - "Check In"

**Request 14 - Search \***

**Send:**

search\_type (all, specialties, map (people, members, locations), organizations, links, specialty\_detail, locations, pending removals, approved removals)

**Receive:**

Search results (categorized?)

**When/Where:**

Every Screen implements one of the search types

**Request 15 - Create Person**

**Send:**

person's info

**Receive:**

created person's id

**When/Where:**

Screen 13 - on Submit

**Request 16 - Create Specialty**

**Send:**

specialty info

**Receive:**

created specialty's id

**When/Where:**

Screen 13 - on Submit

**Request 17 - Create Organization**

**Send:**

organization info

**Receive:**

created organization's id

**When/Where:**

Screen 13 - on Submit

**Request 18 - Create Link**

**Send:**

Link info

Link location (top\_level, agency\_id, location\_id, member\_id, person\_id, specialty\_id)

**Receive:**

created link's id

**When/Where:**

Screen 13 - on Submit

**Request 19 - Create Organization Location**

**Send:**

organization\_id

location info

**Receive:**

created location's id

**When/Where:**

Screen 13 - on Submit

**Request 20 - Edit Specialty**

**Send:**

specialty\_id

specialty info

**Receive:**

success or fail

**When/Where:**

Screen 14 - on Submit

**Request 21 - Edit Organization**

**Send:**

organization\_id

organization info

**Receive:**

success or fail

**When/Where:**

Screen 14 - on Submit

**Request 22 - Edit Link**

**Send:**

link\_id

link info

**Receive:**

success or fail

**When/Where:**

Screen 14 - on Submit

**Request 23 - Edit Person/Member**

**Send:**

member\_id or person\_id

member/person info

**Receive:**

success or fail

**When/Where:**

Screen 14 - on Submit

**Request 24 - Edit Organization Location**

**Send:**

location\_id

location info

**Receive:**

success or fail

**When/Where:**

Screen 14 - on Submit

**Request 25 - Edit Flagged Item**

**Send:**

flag\_id

flag info

**Receive:**

success or fail

**When/Where:**

Screen 14 - on Submit

**Request 26 - Edit Role**

**Send:**

role\_id

role info

**Receive:**

success or fail

**When/Where:**

Screen 14 - on Submit

**Request 27 - Flag For Removal**

**Send:**

flag info (agency\_id, location\_id, link\_id, member\_id, person\_id, specialty\_id)

**Receive:**

flag\_id

**When/Where:**

Screen 14 - on Submit