Roomem readme

DATABASE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This section serves as an outline of the structure of the Roomem MongoDB database. All the connection parameters are indicated within.

NON-AGENT SUBMISSION

When a roomie (non-agent) user creates a profile, they either already live in an apartment or they don╒t. This is reflected in the option they pick when on the landing page:

\* ╘Find a Roomate╒ option: a user looking for a room will a have a Boolean ╘hasRoom╒ set to 0 or false upon form submittal. Their ╘roomID╒ parameter will also be set to ╘none╒, since they do not have a room to point to in the database.

\* ╘List a Room╒ option: a user listing a room will have ╘hasRoom╒ set to 1, and the room parameters they enter into the form will generate an entry in the ╘room╒ collection. This ╘room╒ entry will have a unique ID which will be returned, and the roomie╒s ╘roomID╒ parameter will be set to it.

DB PARAMETERS

DB Name: roomemDB

Collections:

╘roomies╒ ╨ This collection is strictly for non-agent individuals. It contains the data of all individuals ╨ with or without a place ╨ who are looking for a roommate. As specified above, ╘roomies╒ with a place will have their ╘hasRoom╒ parameter set to 1, and their ╘roomID╒ parameter will be set to the unique \_ID of the room in the ╘room╒ collection. ╘Roomies╒ without a place will have ╘hasRoom╒=0 and ╘roomID╒=╒none╒. Parameters are:

- \_ID (generated by Mongo)

- created (String ╨ keeps a record of entry creation time)

- hasRoom (String ? Number)

- roomID (String)

- couple (Number)

- name (String)

- gender (String)

- age (number)

- hometown (String)

- hours (String)

- industry (String)

- pets (String)

- smoke (String)

- stuff (Number ? String)

- budget (Number)

- neighborhood (String)

- bio (String)

- hasApplied (String)

- match:

{

- gender (String)

- couple (Number)

- ageMin (Number)

- ageMax (Number)

- hours (String)

- industry (String)

- pets (String)

- smoke (Number)

}

╘rooms╒ ╨ This collection is strictly for rooms, whether posted by an agent or by a roomie. This discriminant can be determined by the Boolean ╘listedByAgent╒, which is set to 1 if the room was listed by an agent or to 0 otherwise. When a room is submitted by either roomie or agent, it generates an entry in this collection and, as specified above, the unique ID Mongo generates is returned to set the roomie╒s or agent╒s ╘roomID╒ parameter. The lister╒s own unique ID is also entered into this collection as ╘listerID╒. Parameters are:

- \_ID (generated by Mongo)

- listingName (String)

- byAgent (Number)

- listerID (String)

- price (Number)

- numOfRooms (Number)

- numOfRoommates (Number)

- stuff (String)

- securityDeposit (String)

- about (String)

inclusions:

{

- hotWater (Number)

- electricity (Number)

- laundry (Number)

- elevator (Number)

- backyard (Number)

- rooftop (Number)

- balcony (Number)

- privEntrance (Number)

- parking (Number)

- gym (Number)

- desk (Number)

- handicapAcc (Number)

}

╘agents╒ ╨ When an agent lists any number of individual rooms, it or they are part of a completely vacant apartment. Therefore, an agent will have several room unique ╘roomID╒s returned by the ╘room╒ collection.

(Note that all the above criteria that can be the result of choosing from a dropdown menu can be represented by integers (Number) since they represent a finite and determined pool of options. For instance the hours option would show up in the form as a dropdown with ╘9to5╒, ╘night╒, and ╘graveyard╒ as options. These options could be imported into the backend as ╘0╒ and ╘1╒, and ╘2╒ respectively to avoid the processing of lengthy strings.)

DB INTERFACE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

An interface subsystem of routes and controllers can be created to test the modification and DB elements for testing and maintenance purposes. This would be reachable through the roomem/db\_util route.

This could also server as a precursor for the interface that an agent can use to manage properties he/she has advertised.

Note:

- complete the contacts app

- migrate db integration to mongoose

- change name of folder to roomemdb

- change Angular controller names to relevant names

- change connection to roomemdb

- gradually change html file to reflect real roomem params

- separate into folders as shown in the vertical model in MEAN book

- check what happens when you enter a partial roomie

- see if html can be split so that Angular can pull in views

- set defaults for all match values so that entries can be taken with blank fields

- test an ng-view

- split current DB Query into three views (sign in, primary, match) to test cross-view ID retention and use using sessionStorage

- make a new controller for this sign in process instead of bloating the one

that already exists for DB Query ╨ can still be in the same Angular module

Also make a new routes file for it. Rename the old ones db query

- Test dude╒s view rendering method

- See if it can be implemented in our app

- Integrate our ready views into our working app

- Begin to integrate the angular model into the actual user views we created

- Create the user login method

- Get room dbQuery CRUD working

- Get dbQuery CRUD views to add rooms as a user

APIs \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GOOGLE MAPS API KEY:

AIzaSyBS7LbBh49rHVI2qdrZvACM-PqPVIYCHzE

Facebook API key (for oAuth)

App ID: 869224333191135

App Secret: 1c121059bb25ffdf44cf1b65e3a67405

FB test app : 905182259595342

Cloudinary API key:

Cloud name: qpalzmwoskxn

API key: 996189833629334

API Secret: fphFqi-6VRXepp718d-7aCmpSGw

Environment variable: CLOUDINARY\_URL=cloudinary://996189833629334:fphFqi-6VRXepp718d-7aCmpSGw@qpalzmwoskxn  


Rev\_6: 12.01.2015: Created a service for modals. Modals can now be used to sign up new users. The error messages are also sent back to the client, so they just have to be put into the markup so the user can see them. No Fabcebook OAuth has been worked yet.

Rev\_10: 12.10.2015: Created a working matching algorithm. In the view, the scores are show as percentages in descending order. Got rid of unused views and controllers. Changed the matching weights so that “don’t care” returns a 1 instead of a zero. The rest of the weights got bumped up by one too. Put a dev and regular menu on every view.

Rev\_11: the roomies field in the room model stores the listerID. There is a field in the room profile page that allows the user to dynamically add fields for additional roomie emails. The addRoom() button calls the fn in the room controller which takes the room’s fields, and also builds an object containing the added roomies emails, generates a random string password, and creates a new user for each email, returning the ID of each. It then posts the room fields in a new room entry. Theoretically, the new roomie IDs should be pushed onto the roomie field in the new room, but async issues are arising.

Rev\_12: 12.15.2015: The rooms’ roomie field is now successfully populated with additional roomie IDs supplied by the lister roomie. For now, this is being done using a not-so-elegant timeout function, but hopefully will soon be handled by promises. Now, the roomsearch() fn has to successfully query the names of the added roomies when called.

Rev\_13: 12.15.15: The roomSearch() fn now successfully retrieves the names of each roomie attached to a each room returned by the search. The names also display correctly in the room search view. The listerID of the room has also been changed to a DBRef ObjectID type referencing the roomie model (the model being referenced will have to be ‘agent’ if byAgent is true. The byAgent field has to be changed to a Boolean from a String - for future revs.) This is done so that if the byAgent field of a room is true, the name of the lister as returned by the populated listerID should not appear in the view as a roomie. Next, views should be populated dynamically instead of being hard-coded as they currently are.

Rev\_14: 12.17.15: A new factory UpdateSvc has been added to make on-the-fly changes to a room or roomie document. It successfully updates roomie statuses from 2 for a lister adding a room and 3 for an annexed roomie being processed in the AddRoomie service.

Rev\_15: 12/27/2015: Greeting banner works at the top of the page. It greets the logged-in user properly after a successful auth. Next steps are:

* get the Cloudinary API working to handle client-side img uploads
* finish the messaging service so that replies can happen and messages can be announced when they are from someone else than you. Get the single-thread view working from a sliding panel.