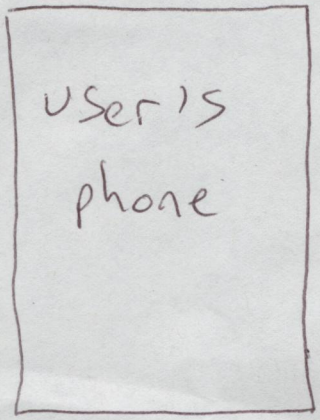
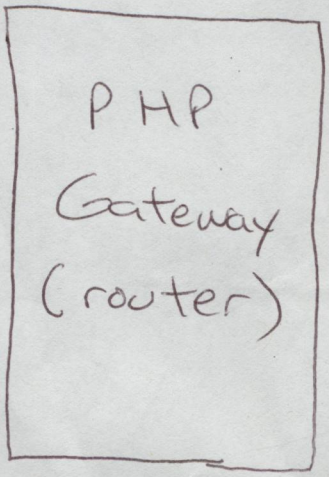
# Send messages General overview

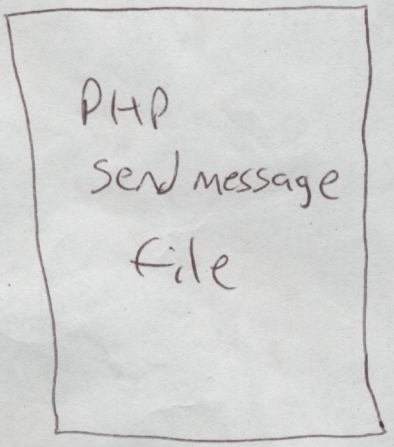



to Server




1) User chooses contact(s)
2) Types message
3) Hits send
4) Goes to /sendmessage route via AJAX

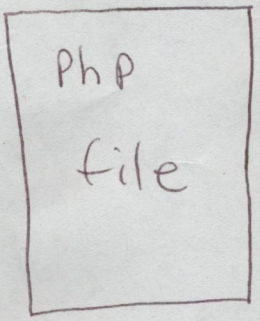
1) router routes message to php file for send messages



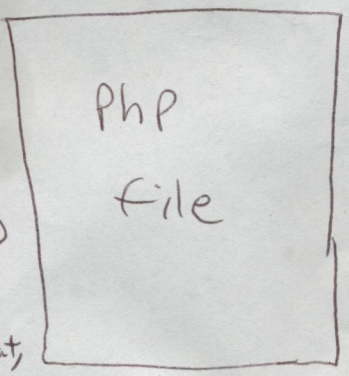

1) Send messages PHP file checks ~~if~~ this is new message thread based on conversation ID

New message thread

Old message thread




1) contacts PHP backend to create ~~a new~~ a new, unique (not duplicated) conversation ID + creates an Empty redis hash for that ID.



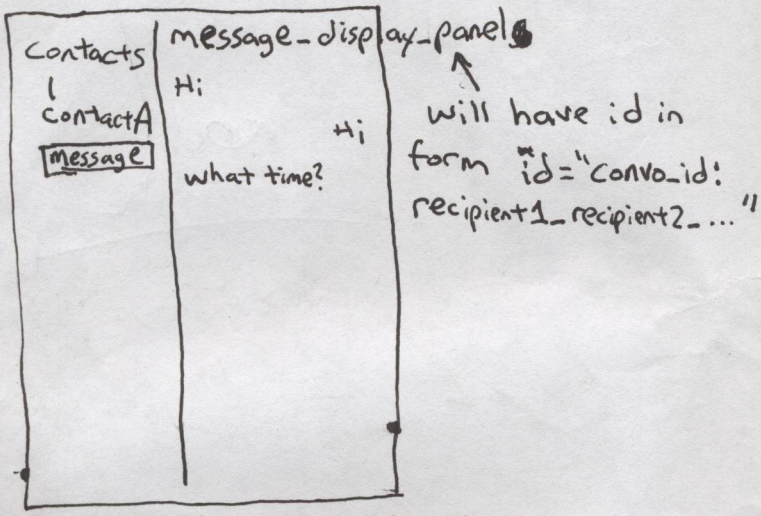

1) locates redis conversation hash ~~with same~~ based on conversation ID + stores message
2) finds redis hash belonging to ~~each~~ recipient of this message

3) sends back, in JSON format, conversation ID + message_sent confirmation_ID as Ajax response

+ puts message in unsent sub hash

recieves AJAX response

# Send messages details




Once user starts a new message, you

1) check localstorage for a recipient list that matches exactly the recipients the user is contacting, if yes, retrieve conversation ID.

2) If No exact match of recipients or no conversation ID exists, even if exact match of recipients, check all message_display_panels' ids in localstorage to see if matching recipients exist. If yes obtain conversation ID, which is also part of message_display_panel id. (See above image)

3) if still no conversation Id found, conversation ID will be ∅.

4) Create a random 8 character string consisting of numbers, letters (uppercase considered different from lowercase) and the symbols "!@_" This will be known as the message-sent_confirmation_ID.

5) Store this Id locally in an array of such Ids.

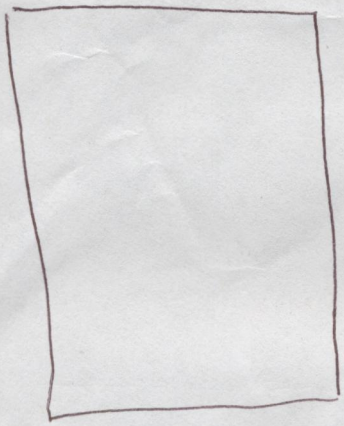
6) create the JSON object to be sent to back-end. it will contain: The conversation ID (∅ if new conversation) An array of the recipients, the message, and the message_sent_confirmation_id

7) send JSON object to back-end route: /sendmessage via AJAX.

8) Wait to receive back message_sent_confirmation_Id in AJAX response from back-end.

9) display something indicating message was sent.

get messages
~~get~~ overview



to server

PHP Gateway (router)

1) a ~~web worker~~ Javascript web worker sends a request via AJAX to the ~~/check for new messages~~ /getmessages route every 100 milliseconds after last check is completed

2) the request will be a JSON object which will have the user's handle

1) routes to php file for ~~check for new messages~~ getmessages

PHP file

AJAX Response

1) Parse JSON ~~R~~response object ~~by convo~~ by conversation ID + sender + ~~then~~ display messages.

1) receives JSON object + gets the users handle

2) checks in redis for the ~~users handle~~ hash with the key as the user's handle + checks the unsent sub hash

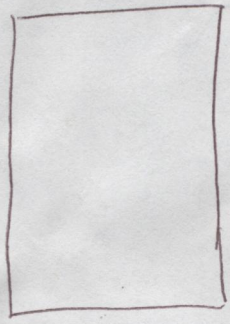
3) grabs all the messages based on conversation ID + timestamp of all unsent messages

4) creates JSON object from all unsent messages

5) sends JSON object as AJAX Response

# Add new contact overview

user A



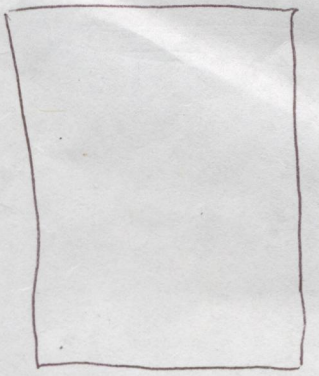
1) In add new contact input field, puts handle of user B

2) ~~sends JSON AJAX request~~ gets location of this user (user A)

3) ~~creates~~ same as below but with handle of user B as new contact

within 500 yards of each other

user B



1) ~~Inputs~~ same as above except puts handle of user A instead.

2) ~~G~~ets location of this user (user B)

3) creates JSON object with handle of contact to add (in this case user A) + with location Data + with current user's handle (user B)

Sends AJAX request w/ JSON object to /addnewcontact route

PhP Gateway (router)

1) routes to addnewcontact php file

php file

1) In Redis new contacts hash adds handle ~~name~~ of current user, location, + handle of contact to add

2) looks for handle of contact to add in same hash, if not found repeats this step.

3) once found will check distance of two users. If within 500 yds, next step.

4) creates a random 6 character pin, stores in redis

both if users enter 6 character pin, they match + get added to each others contacts.

Sends AJax Responce w/ 6 character pin

Db tables - MySQL

## users

(int) id | (String) handle | ~~(String)~~ (String) phone_number | (String) carrier | (boolean) verified | (datetime) ~~timestamp~~ time_date_signed_up | (datetime) time_date_verified

(datetime) ~~date_tim~~ time_date_last_message_sent | (datetime) time_date_last_message_read | (string) long_id

## conversations_overview - *this table content never deletes.

(int) id | (string) long_conversation_id | (JSON formatted string) members {members: [ids, ids, ...], initiator: id, last_timestamp: } | (timestamp) last_timestamp

(datetime) conversation_started_time

## conversations_communications

~~(int) id~~ ~~(int) conversation_id~~ ~~long conversation_id~~ (int) - foreign key short_conversation_id | (string) long_conversation_id

~~A series of JSON objects~~ ~~messages~~ (a comma separated series of JSON objects stringified)

messages

{timestamp: {short_conversation_id, long_conversation_id, sender_id, ~~sender~~ sender_handle, recipients: [ids, ...], message_sent_confirmation_code },

{timestamp: ... }

message ID = long_conversation_id + ":" + timestamp.

## users_current_time_zone:

~~User id~~ (int) foreign key user_id | (string) timezone

## users_contacts

id | user_id | contacts (JSON encoded string)

{group: ... ; contacts: { }}

# Services:

## Front-end:

### UI-service

creates the panels, buttons, + input fields + displays messages.

### send_message_json_service

retrieves message data + packages it as JSON object + sends to back-end (/sendmessage) route via AJAX service

### retrieve_message_json_service — S

continually polls back-end every 100 milliseconds for new, unread messages + receives them as JSON object + sends to UI_service to display. Gets messages from /getmessages route + is a web worker.

### create_random_number_service:

creates 8 character message_sent_confirmation_id random id

### geolocate_user_service: - S

get long, lat of user

### AJAX_Service

as a parameter send data to a specified route via method (also a parameter),

### Sign_up_service

~~retrieve~~ get phone number, handle, + service provider + send to back-end via AJAX_service

# Services:

## Back-end:

### routing_service

routes incoming data to appropriate file to process

### create_random_number_service

### send_text message_to_phone_service

### Pull_from_Redis_and_Save_to_MySQL_service

### convert_data_to_appropriate_format_service

takes incoming JSON data & converts it to appropriate string data that can be saved in redis. Also takes redis data & converts it to ~~JSON to~~ appropriate format to be sent over http request & converted to JSON on front-end.

— via Save_to_redis_service

### Save_to_redis_service

### read_from_redis