Specification of the chat protocol for

`GradAgenda`

0. Meta

- Author: Pengyu CHEN
- Contact: pengyu[at]libstarrify.so
- Changelog:
    - 2014.07.27 Initial version.

1. Overview

The chat protocol is built on the WebSocket protocol, which provides realtime duplex communication. A testing server has been deployed at [TBD]. For now it's a regular WebSocket connection, however in the production deployment it shall be tunneled over TLS.

The payload size of each data frame shall not exceed 4KB. The payload data shall have type "text" and be encoded as UTF-8. The payload data shall be in valid JSON format. Otherwise the server would close the socket with appropriate status codes.

2. The Communication Model

The JSON objects of payloads represent actions between the server and the client. Unlike HTTP's request-response model, our protocol suggests that the server and client shall be implemented in an event-driven model.

2.1. Authentication

After the connection has been established, the server turns into standby mode and waits for the client to authenticate itself.

No other actions shall be sent before a successful authentication action. No other authentication actions shall be sent after a successful authentication action.

2.2. Session Based Messaging

There are two types of sessions:

- Direct sessions: direct chat between two users. Immutable.
- Mutable sessions: also known as "group chats". Can be created, modified, and deleted.

Once a session is created, users could send message to the session and all involved shall

receive a broadcasted message.

3. Action Specification

3.1. Common Data Types

- `timestamp`: *Milliseconds* since the UNIX Epoch.
- `CDID`: Client defined identifiers. String. Used to simulate a request-response cycle in event-driven models. After obtaining a `CDID` from a request, the server would use it for nothing but to attach it to the corresponding response if any.
- `message-type`: Shall be valid JSON. Required fields:
    - `"type"`: `"text"`. (More to be added in the near future.)
    - `"body"`: The message content.

3.2. Client-side Actions

3.2.1. Authentication

Performs authentication. Required fields:

- `"cdid"`: `CDID` as described above.
- `"type"`: `"authenticate"`
- `"user-id"`: User ID of the client.
- `"device-id"`: Device identifier. Shall be unique among devices while persist across logins on a same device, e.g. "device-model&MAC-address".
- `"token"`: Authentication token obtained when signing-in.

The server would send a response action to indicate the authentication results.

3.2.2. Requesting Direct Session Info

Request information of a direct session. Required fields:

- `"cdid"`: `CDID` as described above.
- `"type"`: `"request-direct-session-info"`
- `"user-id"`: ID of the user to chat with.

On failure the server would send a response action to indicate the authentication results. On success the server would soon send a providing-session-info action.

3.2.3. Requesting Mutable Session Creation

Request creation of a mutable session. Required fields:

- `"cdid"`: `CDID` as described above.
- `"type"`: `"request-mutable-session-creation"`
- `"user-id"`: Array of the user ID of participants.

On failure the server would send a response action to indicate the authentication results. On success the server would soon send a providing-session-info action.

3.2.4. Requesting Mutable Session Modification

Request modification of a mutable session. Required fields:

- `"cdid"`: `CDID` as described above.
- `"type"`: `"request-mutable-session-modification"`
- `"session-id"`: ID of the session to be modified.
- `"user-id"`: Array of the user ID of participants.

On failure the server would send a response action to indicate the authentication results. On success the server would soon send a providing-session-info action.

3.2.4. Requesting Mutable Session Deletion

Request modification of a mutable session. Required fields:

- `"cdid"`: `CDID` as described above.
- `"type"`: `"request-mutable-session-deletion"`
- `"session-id"`: ID of the session to be deleted.

On failure the server would send a response action to indicate the authentication results. On success the server would soon send a providing-session-info action.

3.2.5. Requesting Mutable Session Listing

Request listing of mutable sessions that the client is in. Required fields:

- `"cdid"`: `CDID` as described above.
- `"type"`: `"request-mutable-session-listing"`

The server would soon send several providing-session-info actions.

3.2.6. Requesting Sending A Message

Request sending a message to a session. Required fields:

- `"cdid"`: `CDID` as described above.
- `"type"`: `"request-sending-a-message"`
- `"session-id"`: ID of the session to be sent to.

- `"message"`: `message-type`. Message to be sent.
- `"sender-timestamp"`: `timestamp`

On failure the server would send a response action to indicate the authentication results. On success the server would soon send a message-broadcast action.

3.3. Server-side Actions

3.3.1. Status Response

Giving responding status of a specific client action. Fields:

- `"cdid"`: `CDID` as described above.
- `"type"`: `"status-response"`
- `"status"`: `"succeeded"` | `"failed"`
- `"message"`: (optional) An message describing the response.

3.3.2. Providing Session Info

Providing information of a session. Fields:

- `"type"`: `"providing-session-info"`
- `"session-id"`: ID of the session.
- `"session-type"`: `"direct"` | `"mutable"`
- `"status"`: `"created"` | `"modified"` | `"deleted"` | `"unchanged"`
- `"user-id"`: Array of the user ID of participants.

For any two users the ID of their direct session shall never change. Thus the client could use some cache. For direct sessions the `"status"` shall always be `"unchanged"`.

3.3.3. Message Broadcast

Broadcasting message from a session. Fields:

- `"type"`: `"message-broadcast"`
- `"session-id"`: ID of the session the broadcast is sent from.
- `"sender-id"`: ID of the user the message is sent from.
- `"message"`: `message-type`. Message to be sent.
- `"sender-timestamp"`: `timestamp`

3.4. Batch Mode Actions

Either the server or the client may send a batch mode action instead of a series of individual actions. The effect of the batch mode action shall be equivalent to sending the individual actions one by one. Required fields:

- `"cdid"`: `CDID` as described above. Only required when the action is sent by the client.
- `"type"`: `"batch-mode-action"`
- `"actions"`: Array of the individual actions.

4. Future Updates

The protocol is expected to be updated soon, with backward compatibility, of course. Some of the features that might be potentially supported are:

- Multimedia messages, e.g. images and voice clips.
- Explicit message history requesting.
- Delivery reports and read reports like it in Google Hangouts and Facebook Chat.
- Group chat access control