Routes:

An error response will always be of the form:
{"success": False, "error": message}
The following routes assume an api call goes through successfully.

1. POST /api/signin/

Request: `
{
"username": username <string>,
"password": password <string>,
“first_name”: firstName <string>,
“last_name”: lastName <string>
}
Response:
{
"success": True,
"data": {
“user_id": userId <integer, the id associated with this user>,
"username": username <string, the username of this user>,
“first_name”: firstName <string>,
“last_name”: lastName <string>
}
}

Eg: Request json is going to look like
{“username”: “ab123”, “password”: “pwd”, “first_name”: “Jack”, “last_name”: “Smith”}
Response is pretty straightforward and is not really important because we won’t be using any of these further. The “data” would look something like
{“user_id”: 1, “username”: “ab123”, “first_name”: “Jack”, “last_name”: “Smith”}

2. POST /api/login/

Request:
{
"username": username <string>,
"password": password <string>
}
Response:
{
"success": True,
"data": {
“user_id": userId <integer, id of this user>,
"username": username <string, username of this user>,
"token": access_token <string, unique token associated with this user>
}
}

Calling this is exactly the same as just using the username and password fields of the signing route, except this time it checks if the user actually has an account in the database, an error is reported if the account doesnt already exist. For the response json, you can access the “data” element, within which you can access the “Token” element, and this needs to be stored in a global var probably because we use it throughout the app, I will refer to this variable as simply ‘token’ from here, it basically just gets called in the header for every subsequent route.
To log in the user we just created an account for, request json:
{“username”: “ab123”, “password”: “pwd”}
And the response would be something like “data”:
{“user_id”: 1, “username”: “ab123”, “token”: “u2i34y3927ry823rgyu22983y48u32”}
This string for token gets stored in the token variable.

3. POST /api/newChat/

Request:

```
{
"from": fr_id <integer, the userId of the user sending the message>,
"to": to_id, <integer, the userId of the user receiving the message>
}, { headers: { Authorization: "Bearer " + token } }
```

Response:

```
{
"success": True,
"data": {
  "from": fr_id <integer>,
  "to": to_id <integer>,
  "from_channel": fr_channel <string, channel of user sending msg>,
  "to_channel": to_channel, <string, channel of user receiving msg>
  "channel_name": chat_channel, <string, channel of the 2 users>
}
}
```

Suppose we have another user: {“user_id”: 2, “username”: “jk345”, … } This route checks if a channel between these two users already exists, and if it does not creates it. The request is going to look like
{“from”: 1,”to”: 2},
{headers: {Authorization: "Bearer " + “u2i34y3927ry823rgyu22983y48u32”}}
if we want to send a msg from ab123 to jk345.

The response is going to look like (the “data” part at least)
{“from”: 1, “to”: 2, “from_channel”: “private-chat_1”,
“to_channel”: “private-chat_2”, “channel_name”: “private-chat_1_2”}
These are the channels that we subscribe to.
private-chat_1 and private-chat_2 are channels associated with a user, so when anyone sends a msg to them this is the channel thru which the ‘to’ user finds out. The channel_name is the actual channel where the message gets sent, it is unique for every pair of users in the form of private-chat_1_2 for eg if we want to send a msg from user 1 to user 2. If private-chat_1_2 already exists but we want to send a msg from user 2 to user 1, we obviously use this same channel.

The backend will handle all of this stuff, just request with from and to, and use the channel_name that gets returned.

4. POST /api/authentication/ (don't need to worry about this one atm)

   Request:

```
{
"channel_name": channel_name,
"socket_id": socket_id
}, { headers: { Authorization: "Bearer " + token } }
```

5. POST /api/sendMsg/

   Request:

```
{
"from": fr_id <integer>,
"to": to_id <integer>,
"message": message_content <string>,
“channel_name": channel_name <string, format is
                                        “private-chat_{fr_id}_{to_id}”>
}, { headers: { Authorization: "Bearer " + token } }
```

   Response:

```
{
"success": True,
"data": {
  “message_id": id,
  “message": contents,
  "from": fr,
  "to": to,
  “channel_name": channel_name
      }
}
```

While the newChat route makes sure that channels for communication between the 2 users exist and returns what they are, this route is used for actually sending the message. Once we have created private-chat_1_2 by calling the newChat route, we use this as follows:
Request is going to look like:
{“from”: 1, “to”: 2, “message”: “hello”, “channel_name”: “private-chat_1_2”}
And the data from the response isn't really important, it just packages this and sends it back again.
By 'subscribing' to this channel the client can find out if there has been a change, and if there has it means a new msg has been added, which can be accessed and added to the tableView. As I said, I'll push the swift for doing this.

These two routes just return stuff that's already been described above. I am going to remove the authorization header from these since it just makes things difficult for you guys. You can just call it like a normal route

6. GET /api/getMsg/<int:channel_id>

   Response:

```
{
"success": True,
"data": [
  {
      “message_id": id,
      “message_contents": contents,
      "from": fr,
      "To": to,
      "channel": channel_id
  },
  ... and so on for all the messages in this channel, data would be a list
in this case
]
}
```

7. GET /api/getAllUsers/

   Request:

```
{ headers: { Authorization: "Bearer " + token } }
```

   Response:

```
{
"success": True,
"data": [
  {
      “user_id": self.id,
      "username": self.username
  },
… and so on for all users in the database, data would be a list in this
case
]
}
```