User settings system

Mentorship System

by Bartek Pacia Google Code-in 2019

In this document I will present my idea on how settings functionality could be implemented in Mentorship System, both on server side and client side. I will go from high-level API implementation to how it would look in the database.

Overview

Any non-trivial app has a settings screen, where the user can adjust important aspects of its working to their own needs. Such settings can be divided in many categories, but for the purpose of this article I'll divide them into two types: ephemeral and persistent.
Ephemeral settings are settings which probably should not be applied on all devices where the user has the app installed (i.e they are device-specific). Examples include power-saving mode, default image save location or video autoplay settings. I won't be focusing on this type of settings, because implementation of them varies among different clients and the focus of this document is backend architecture.
User settings, on the other side, should be stored on the server. Why? There are many reasons, but the simplest one is that storing them on user's device creates many strange ambiguity-related problems. I will use an example again: user buys a new phone and opts-in to the newsletter, but on the old phone he's still unsubscribed from it. Which setting should be taken into account when sending a newsletter email? Of course, the one that is on the server.

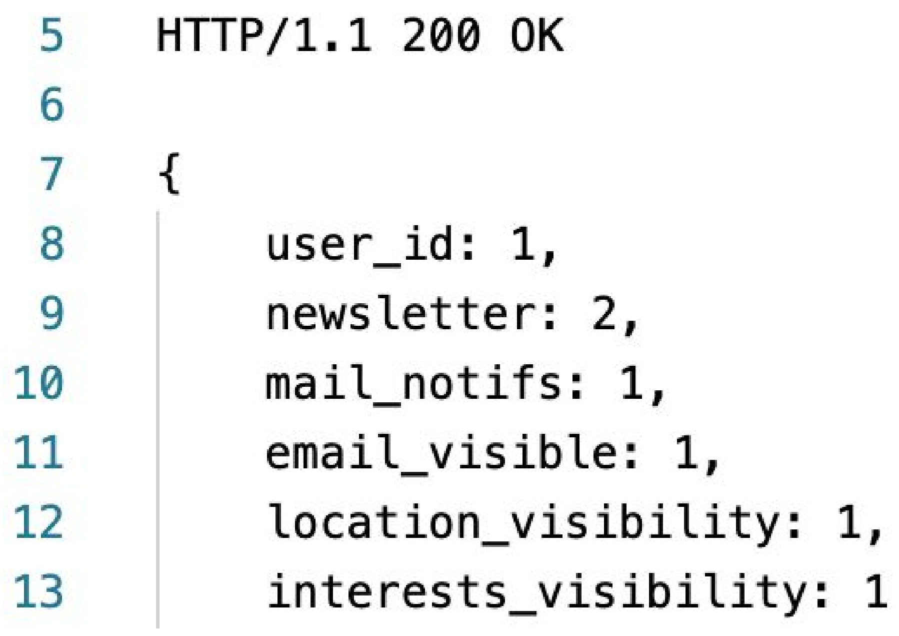
API design

Mentorship System uses REST API for all of its client-server communication. Settings functionality would be no different.
To maintain clear structure, a separate namespace /settings would be created.
In that namespace there would be 2 endpoints:
GET /settings
Gets settings for the user specified by auth token sent with the request.

```
GET /settings HTTP1.1
Authorization: Bearer 1f32d...
```

Figure 1.1 GET /settings example request

Request response is a JSON object containing information about all user’s settings.

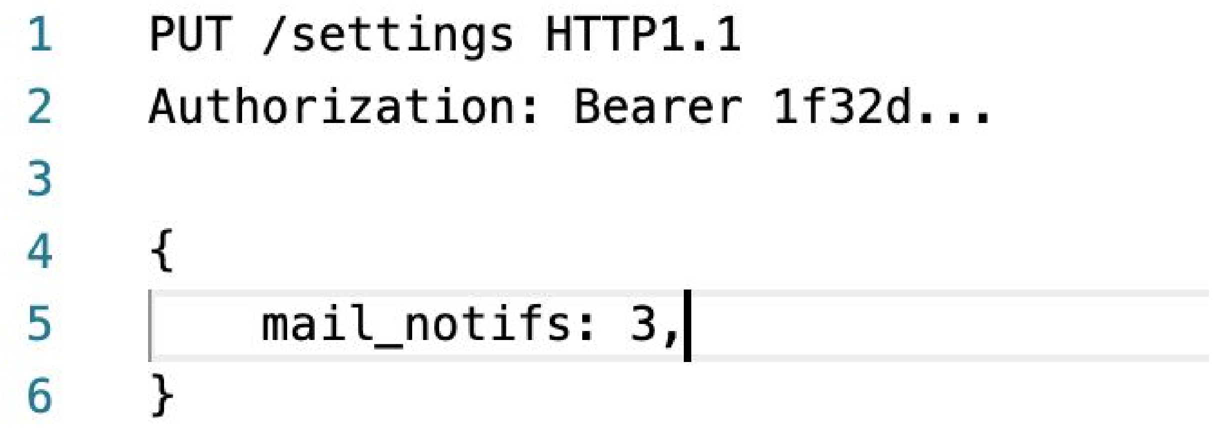


```
HTTP/1.1 200 OK

{
    user_id: 1,
    newsletter: 2,
    mail_notifs: 1,
    email_visible: 1,
    location_visibility: 1,
    interests_visibility: 1
```

Figure 1.2 GET /settings example response

PUT /settings
Updates settings for the user specified by auth token sent with the request.
Request body is a JSON object containing one or more settings to be updated.

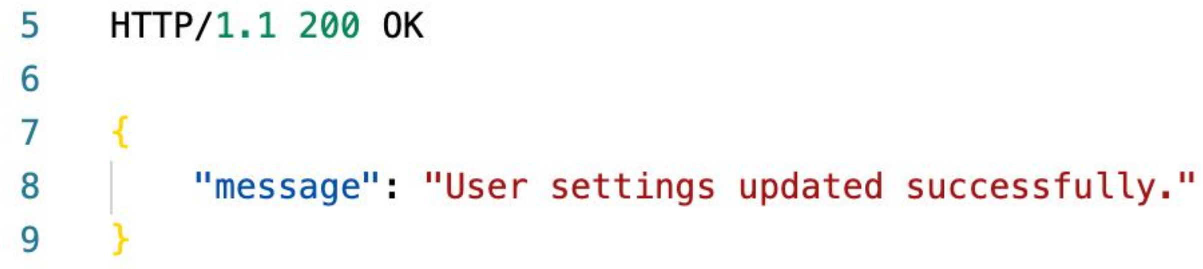


```
PUT /settings HTTP1.1
Authorization: Bearer 1f32d...

{
    mail_notifs: 3,
}
```

Figure 2.1 PUT /settings example request

Response is a JSON object with message field that can be used by the client to inform user about the status of the operation, for example:



```
HTTP/1.1 200 OK

{
    "message": "User settings updated successfully."
}
```

Figure 2.2 PUT /settings example response

Database design

On the lower level, every information is stored in some kind of database. Mentorship backend uses SQLite as a database and SQLAlchemy as Object Relational Mapper to make working with the database easier in Python code.
To store users' settings, I suggest creating a new table, which structure would be following:

Table name: settings

user_id [int] Primary Key	newsletter [int]	mail_notifs [int]	privacy [int]
1	0	2	000...111
2	0	1	000...000
2	3	0	000...333

What settings would the users be able to change? Currently the app is at early stage of development, so there are not many features, but I'll draw some realistic examples:

Newsletter config stored as integer, where:

0 maps to "I don't want to receive newsletter at all."
1 maps to "Send me newsletter occasionally."
2 maps to "Send me everything related to Systers!"

Mail notifications config stored as integer, where:

0 maps to "I don't want to receive email notifications at all."
1 maps to "Send me email notification only when I get new relationship request"
2 maps to "Notify me about literally everything (like tasks being completed by my mentee or new messages (feature not available in the app yet))

Private info visibility stored as integer which acts as a holder for 32 digit decimal number. This is approach similar to the one widely used in Linux for permission management. It would allow user to have granular control over what they share with the world. 32 digits ensure that this solution is future-proof i.e we will we able to add many permissions, and the number being decimal makes it possible to add 10 levels of permission control for every permission.

Number	Permission

0	Only I can see this information
1	Only I and my current mentor/mentee can this information
2	Only I and my current and past mentors/mentees can this information
3	Everybody can this information
... unused [6 more levels can be added in the future]	… unused [6 more levels can be added in the future]

The last digit represents email visibility, the last but one – location visibility and so on, before last but one – interests visibility and so on. Counting from the last digit allows us to add new options without affecting the existing ones.

Example: ...000123 (leading zeros omitted)

- everybody can see this user's email
- current and past mentors/mentees can see their location
- only current mentor/mentee can see their interests

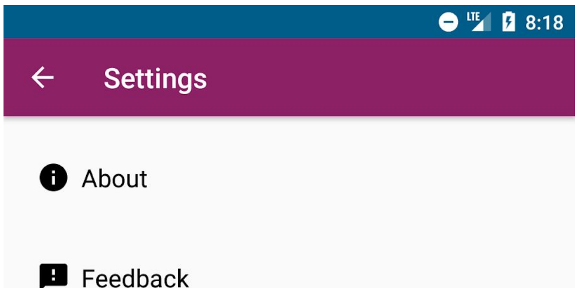
This solution is very lightweight and flexible.

Parsing

JSON object keys are mapped on server side to specific digits of the "privacy" column, so it's easier for clients to use the API.

Applying settings

Last but not least, we probably want to use those settings somewhere so they will actually have an effect.
Most of aforementioned settings would be fairly easy to implement – for example, when an email notification is to be sent to a specific user, system queries database for settings of that user and basing on the settings data retrieved, decides whether to send or not send the response.
Adding user privacy settings would require adding appropriate checks to every Data Access Object. It would be a lot of work, but is definitely doable.



Settings on the client side

Currently, the only official client for Mentorship System is the Android app, so I will describe the flow using android terminology.
I suggest creating a separate UserSettingsActivity. It would show upon clicking a newly added button in SettingsActivity.
UserSettingsActivity would take advantage of native settings functionality provided Android Framework.

Summary

So that's it. I hope my suggestions make sense and will help developer(s) who will implement this important feature. In today's world, privacy is a really hot topic. By having transparent user policy, being compliant with laws like GDPR and CCPA and providing good tools to manage users' personal data, we will be sure that it won't burn us.