Exercise

We've seen 4 different types of HTTP methods that we can use for our endpoints

- GET
- POST
- PUT
- DELETE

which are the most common methods that you'll encounter and use in your daily lives. However, there are also some other, more situational, less common methods. One of those we're going to look at now and it is the PATCH method.

The PATCH method is used when you want to apply partial modifications to a resource. It is similar to the PUT method in that you specify exactly the resource you want to update, but the PUT method is used for either an insert or a replace, whereas the PATCH is a partial update.

The exercise is to implement the PATCH method to update the user information. Specifically, the PATCH method should allow updating both the long_bio and the short_description, but no other properties.

The response body should contain the FullUserPorfile.

Solution

```
class UserProfileInfo(BaseModel):
   short_description: str
   long_bio: str


class FullUserProfile(User, UserProfileInfo):
   pass


def partial_update_user(user_id: int, user_profile_info: UserProfileInfo)
-> None:
    global profile_infos
    short_description = full_profile_info.short_description
    long_bio = full_profile_info.long_bio
    profile_infos[user_id] = {
        "short_description": short_description,
        "long_bio": long_bio,
    }
    return None

@app.patch("/user/{user_id}", response_model=FullUserProfile)
def patch_user(user_id: int, user_profile_info: UserProfileInfo)
-> FullUserProfile:
    if user_id not in profile_infos:
        # return an error - see later lessons :)
        pass
    partial_update_user(user_id, user_profile_info)
    return get_user_info(user_id)
```

Note that we've also updated our FullUserProfile model, and separate the profile info out into a separate class. This way we don't have to create duplicate classes that hold the same information, and instead can just build up our final class using multiple base classes.

As you can see, the implementation is very similar to what we had in our patch, but we just made some minor changes to support this new format. In fact, if we wanted to make our code even cleaner and less redundant, we could also update our create_update_user function to user our new partial_update_user function, like this

```
def create_update_user(full_profile_info: FullUserProfile, new_user_id:
Optional[int] = None) -> int:
    global users_content
    if new_user_id is None:
        new_user_id = len(profile_infos)
    liked_posts = full_profile_info.liked_posts
    users_content[new_user_id] = {"liked_posts": liked_posts}
    user_profile_info = UserProfileInfo(**full_profile_info.dict())
    partial_update_user(new_user_id, user_profile_info)
    return new_user_id
```