Schemas

FastAPI is heavily using Pydantic models to validate request payloads and serialize responses. **FastAPI Users** is no exception and will expect you to provide Pydantic schemas representing a user when it's read, created and updated.

It's **different from your `User` model**, which is an object that actually interacts with the database. Those schemas on the other hand are here to validate data and correctly serialize it in the API.

FastAPI Users provides a base structure to cover its needs. It is structured like this:

- `id` (`ID`) – Unique identifier of the user. It matches the type of your ID, like UUID or integer.
- `email` (`str`) – Email of the user. Validated by `email-validator`.
- `is_active` (`bool`) – Whether or not the user is active. If not, login and forgot password requests will be denied. Defaults to `True`.
- `is_verified` (`bool`) – Whether or not the user is verified. Optional but helpful with the `verify` router logic. Defaults to `False`.
- `is_superuser` (`bool`) – Whether or not the user is a superuser. Useful to implement administration logic. Defaults to `False`.

Define your schemas

There are four Pydantic models variations provided as mixins:

- `BaseUser`, which provides the basic fields and validation;
- `BaseCreateUser`, dedicated to user registration, which consists of compulsory `email` and `password` fields;
- `BaseUpdateUser`, dedicated to user profile update, which adds an optional `password` field;

You should define each of those variations, inheriting from each mixin:

```
import uuid

from fastapi_users import schemas


class UserRead(schemas.BaseUser[uuid.UUID]):
    pass


class UserCreate(schemas.BaseUserCreate):
    pass


class UserUpdate(schemas.BaseUserUpdate):
    pass
```

Typing: ID generic type is expected

You can see that we define a generic type when extending the `BaseUser` class. It should correspond to the type of ID you use on your model. Here, we chose UUID, but it can be anything, like an integer or a MongoDB ObjectID.

Adding your own fields

You can of course add your own properties there to fit to your needs. In the example below, we add a required string property, `first_name`, and an optional date property, `birthdate`.

```
import datetime
import uuid

from fastapi_users import schemas


class UserRead(schemas.BaseUser[uuid.UUID]):
    first_name: str
    birthdate: Optional[datetime.date]


class UserCreate(schemas.BaseUserCreate):
    first_name: str
    birthdate: Optional[datetime.date]


class UserUpdate(schemas.BaseUserUpdate):
    first_name: Optional[str]
    birthdate: Optional[datetime.date]
```

Make sure to mirror this in your database model

The `User` model you defined earlier for your specific database will be the central object that will actually store the data. Therefore, you need to define the very same fields in it so the data can be actually stored.