Normalizing & Validating Data

Making sure data is valid and in the format we expect

What is Validation?

Making sure data meets our minimum requirements

For example:

- Verifying that an email is valid and isn’t taken
- Verifying that a password is a certain length

What is Normalization?

Making sure data is in the format we expect

For example:

- Converting all email addresses to lowercase
- Hashing passwords and remember tokens

Database Normalization is different

Database normalization involves:

- storing data in separate tables and
- referencing a single source

Useful, but not what we are talking about right now

Normalization and Validation are tightly linked

The two are nearly impossible to separate

You need to normalize an email before verifying it is avail
    - Normalize, then validate

You need to validate a PW length before hashing it
    - Validate, then normalize

So far in our app...

Our UserService type contains EVERYTHING!

Reads and writes from DB
    w/ gorm, but still...

Tries to validate data
    if ID == 0 in Delete

Also tries to normalize data
    Hashing passwords and remember tokens

This is error prone

Don’t believe me? Check out our Update() method

It doesn’t normalize passwords, so they never get saved!

How do we fix this?

Splitting our code into "layers"

We are going to split our code into layers to isolate responsibilities

We have done things like this before (we use MVC, remember?)

This is the same idea, but inside of the models package

Warning: This isn't the only design option

There are many ways to design code like this

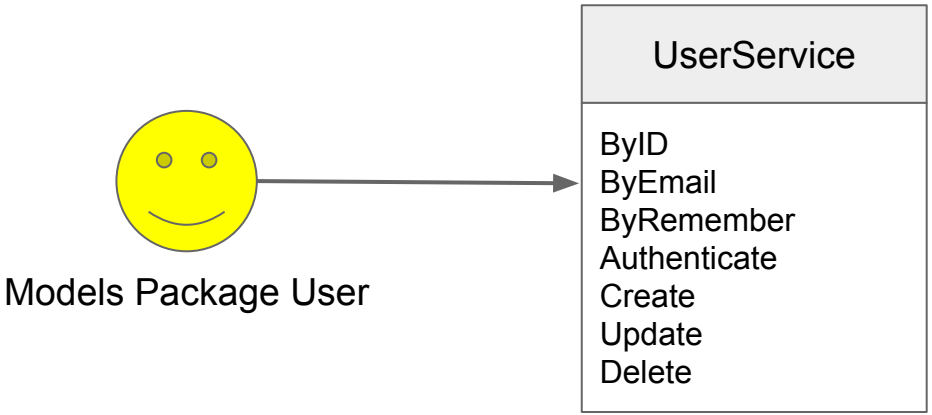
There isn't a "one-size-fits-all" option

I am going to teach you the design I prefer

I suggest you use this your first pass

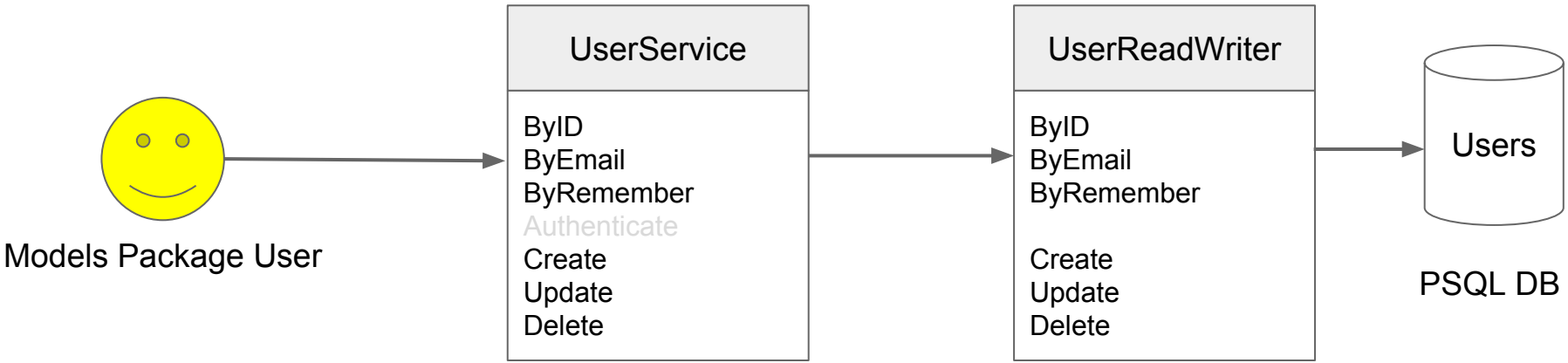
Customize later, when you understand more

The overall design - UserService



Users of our models package will interact with a “UserService” which defines everything they can do with users.

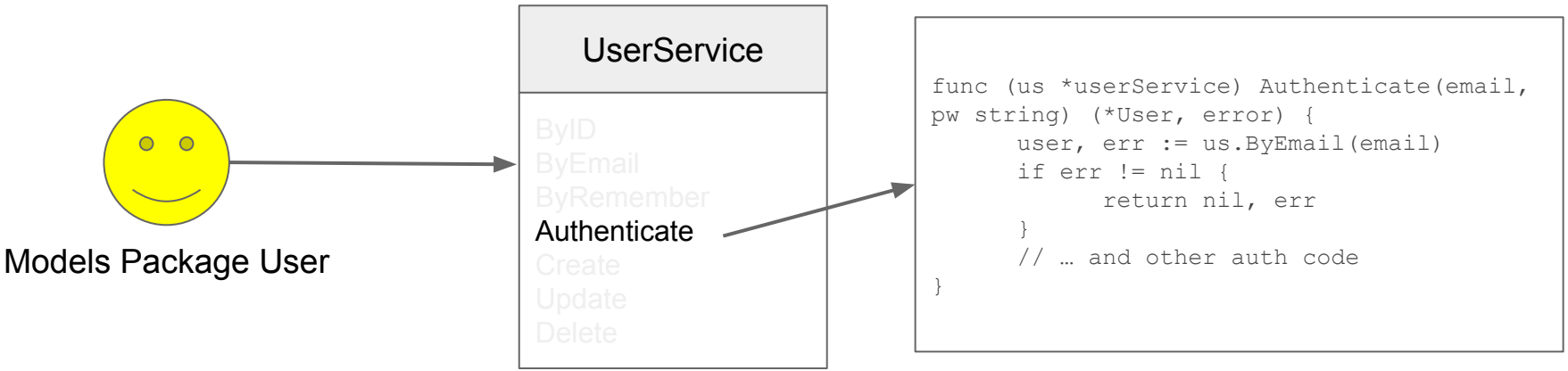
The overall design - Backed by other types



That UserService may use other types to do this work. Our end user doesn't care about that detail.

Eg our DB layer handles reads/writes, but NOT authenticating

The overall design - Or handled by the UserService



Our UserService might handle Authentication itself, or it might defer it to another type.

Again, the end user does not care

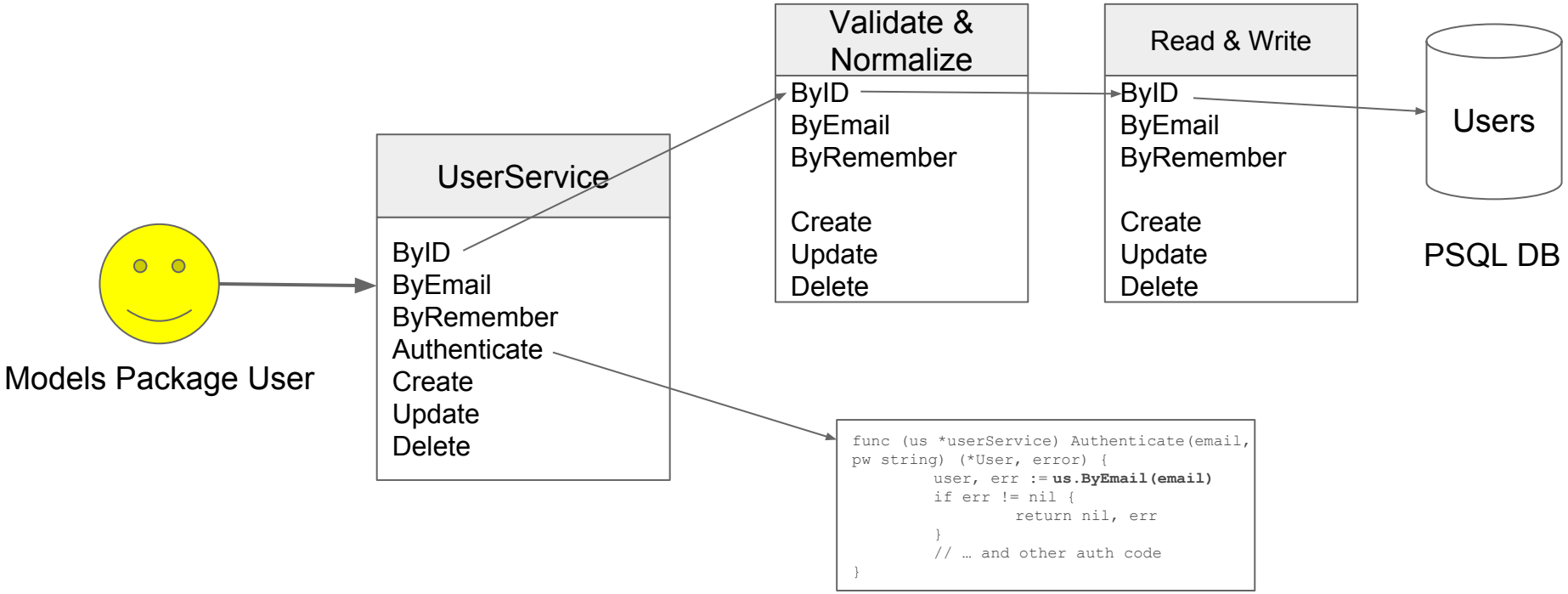
We will start with three layers of responsibility

1. **DB Read/Write** - reading and writing from a DB
   - Doesn't validate or normalize. Simply writes and reads

2. **DB Validation/Normalization** - cleans and verifies data
   - Useful to stick on top of the DB read/write layer

3. **UserService** - top layer and piece we expect users to use
   - Our "contract" to API/package users

What does this look like?



We will be refactoring & implementing all of this!