CSCI4448 Project Writeup

Name: Kyle Helmick

GitHub Link: github.com/Kyle-Helmick/csci4448-loop

Title: Loop

Project Summary

Loop is a fake social media platform for developers. Users sign up with Github and effectively have access to a twitter clone. They can edit their handle, location, and name and make posts to their "page".

Implemented

id	requirements
00	The server authenticates with Github OAuth2
01	The server can access OAuth2 user data returned from Github
02	The server initializes the users' profile using the data from Github in the form of a User object
03	The server creates a document in MongoDB for the user profile under the collection users
05	The user can edit their bio, handle, location, and profile picture
06	The user can make a post
13	The user can logout

Todo

id	requirements
04	The server creates a websocket integrated session for the user that times out at 1 month (prompting the user to sign back in)
07	The user can edit the content of a post
08	The user can delete a post

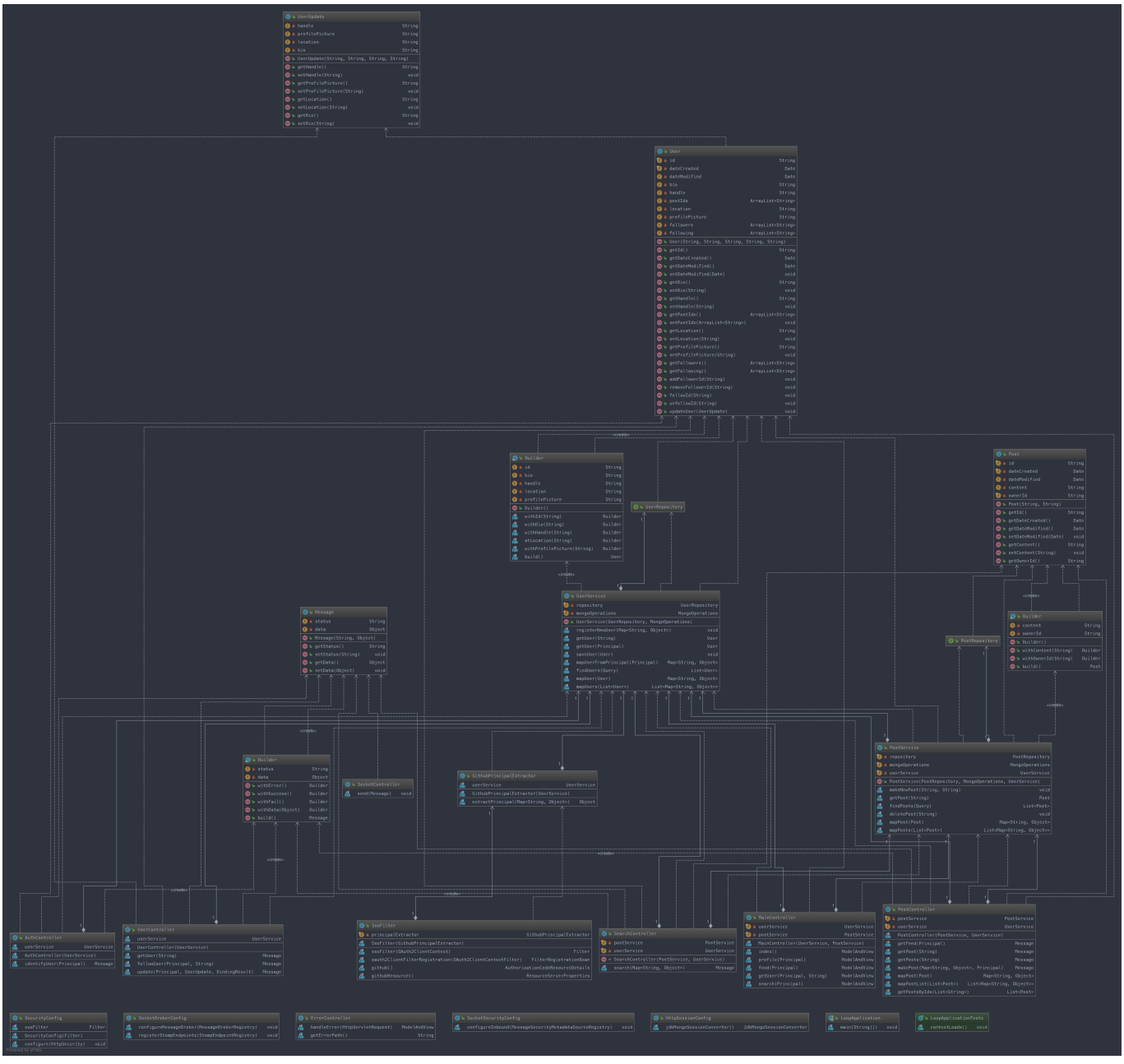
Removed

id	requirements
08	The user can make a group

id	requirements
09	The user can edit the bio and handle of any group that they own
10	The user can delete a group
11	The server will delete any posts or reposts from a group after deletion of said group
12	The group owner can delete any post made in their group

5.

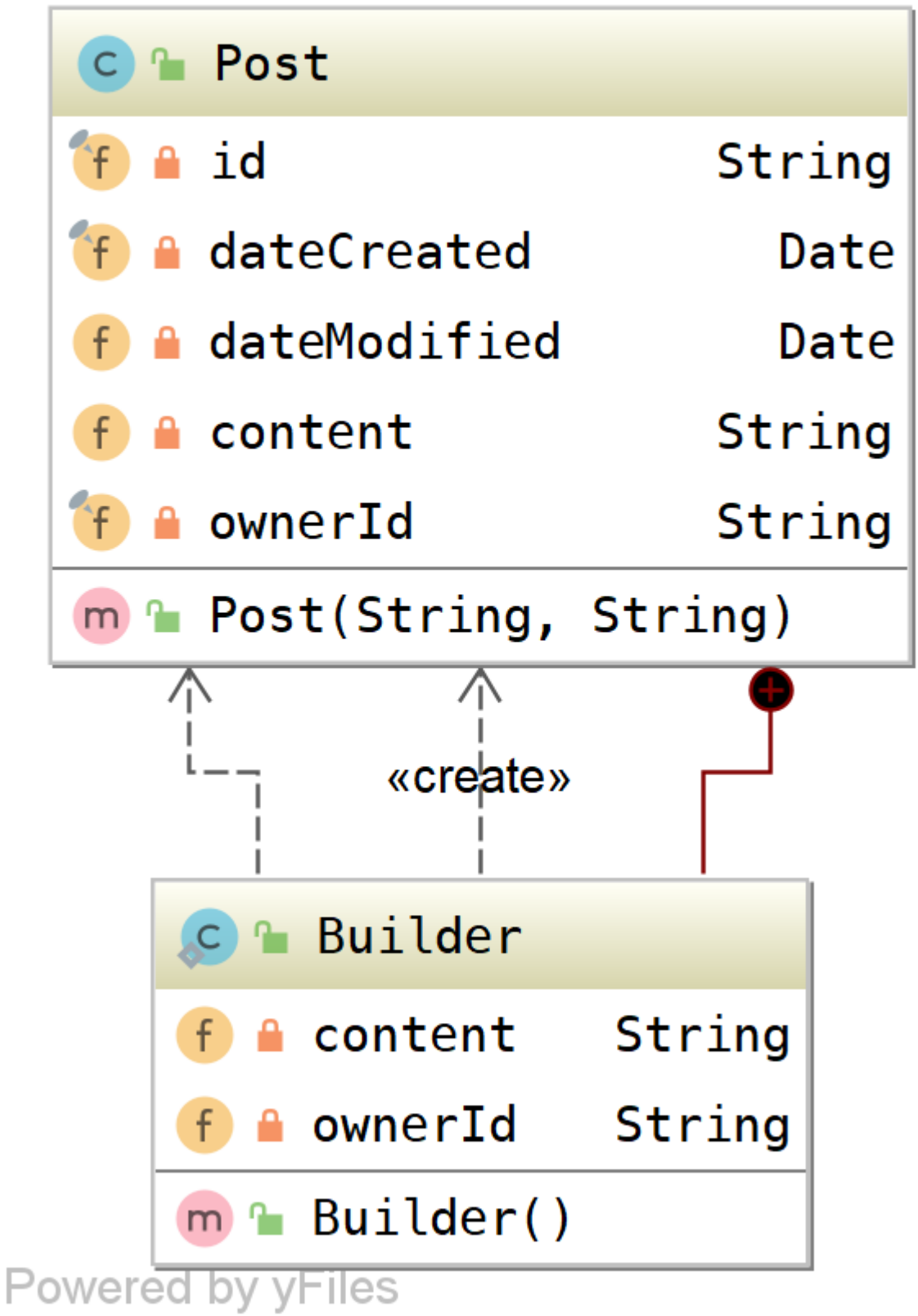
Full Size

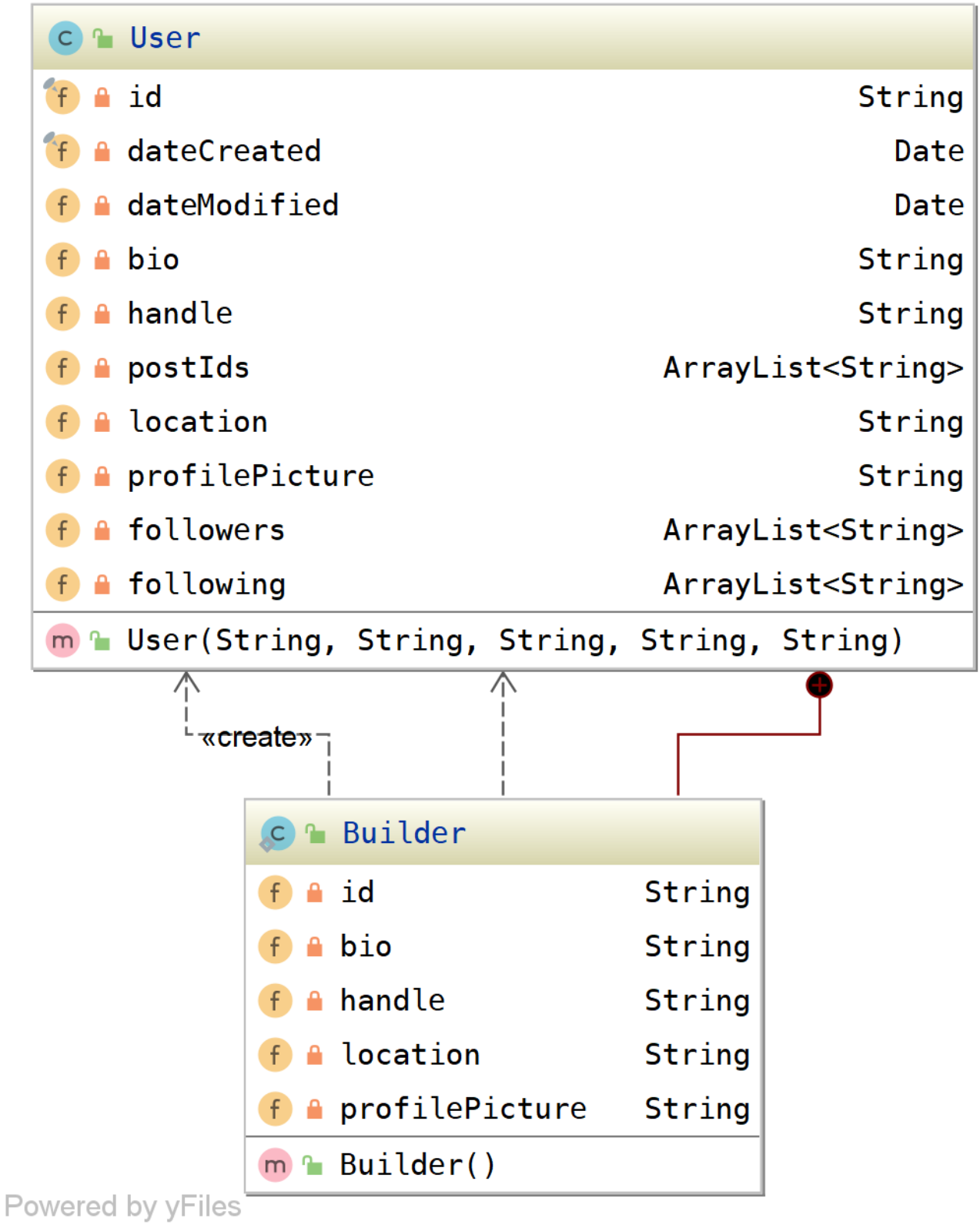


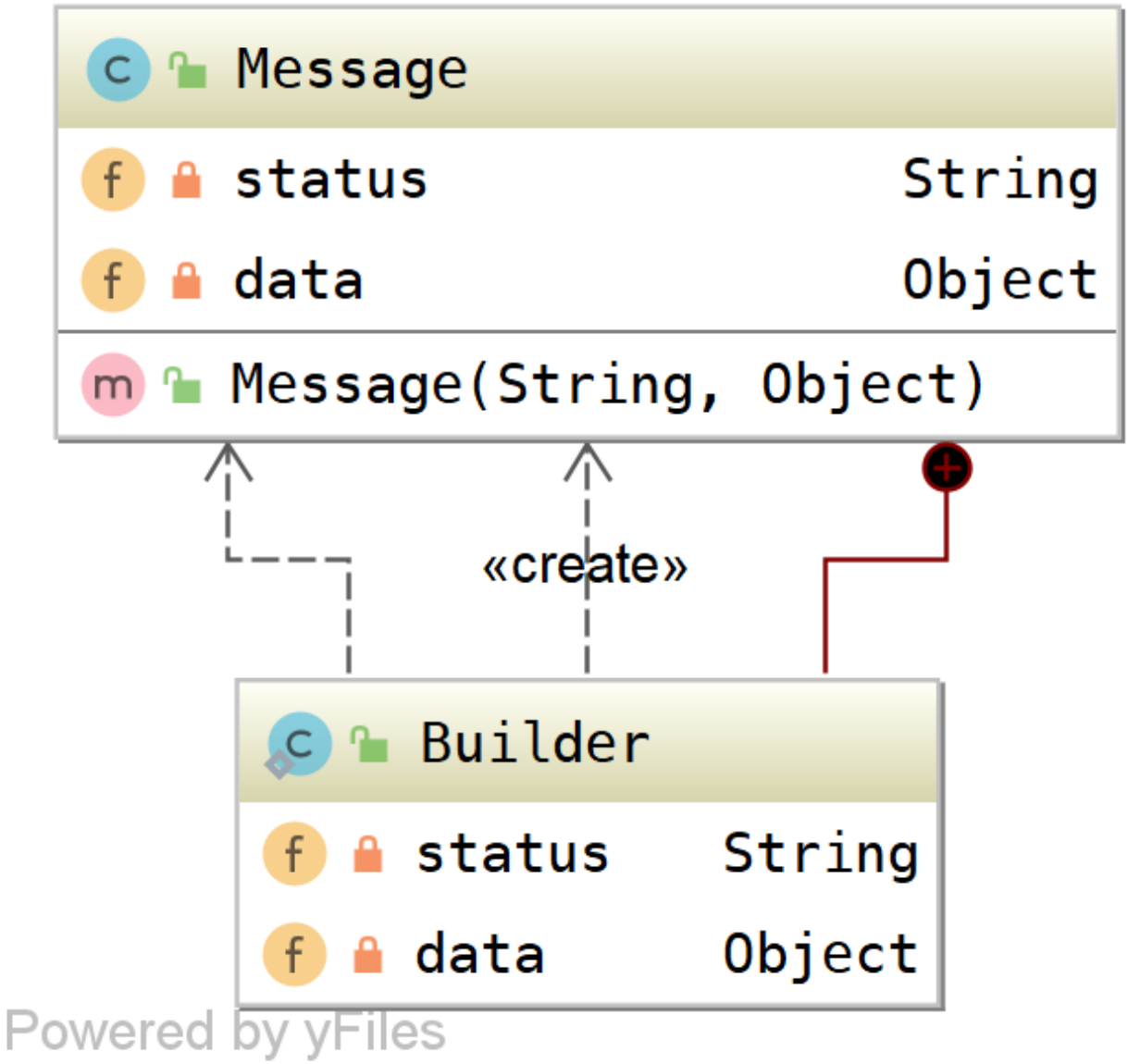
So basically the big difference is that the objects in the models package got bigger (@User) because I dropped the group functionality (overscoped...). I also added more controllers to help make the routing clearer and emphasize the seperation of the API and front facing routes.

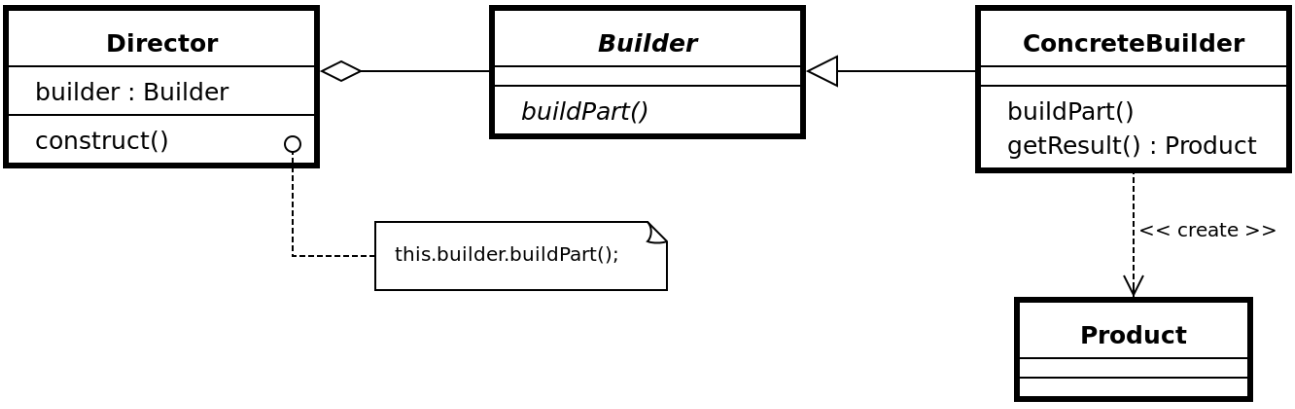
Additionally, my commonly used classes all got builders of their own so the code would be more legible when I'm making those objects. This (Builder) was my design pattern.

6.




User
id String
dateCreated Date
dateModified Date
bio String
handle String
postIds ArrayList<String>
location String
profilePicture String
followers ArrayList<String>
following ArrayList<String>
User(String, String, String, String, String)
«create»
Builder
id String
bio String
handle String
location String
profilePicture String
Builder()
Powered by yFiles





So to implement the design pattern I gave an inner class to each model object called Builder. Builder has custom methods to set the class variables for the class it's declared in and then a method to tie everything together and instantiate the appropriate object. I selected the design pattern because instantiating a new user can be ambiguous as follows:

```
User user = new User(String, String, String, String, String);
```

versus the builder pattern:

```
User user = new User.Builder()
  .withId(String)
  .withHandle(String)
  .atLocation(String)
  .withProfilePic(String)
  .withBio(String)
  .build();
```

Note: "String varName" has been substituted for String to emphasize the abiguity of the signature for the User constructor.

7.

I've learned that it's really good to know the design patterns well and have a good idea of what your model objects will look like before the project. Had I known what the model objects would look like, and had I known Builder before starting I would have saved myself a refactor during the project. Additionally, it's an interesting problem to try and choose the "right" design pattern because there are always quite a few ways to solve a problem. I found myself in that position when trying to decide which design pattern I would try to follow for the project.

Overall I think the project was a lot of fun, but the importance of knowing the design patterns and how to implement them before starting your projects / before solving the problem is key.