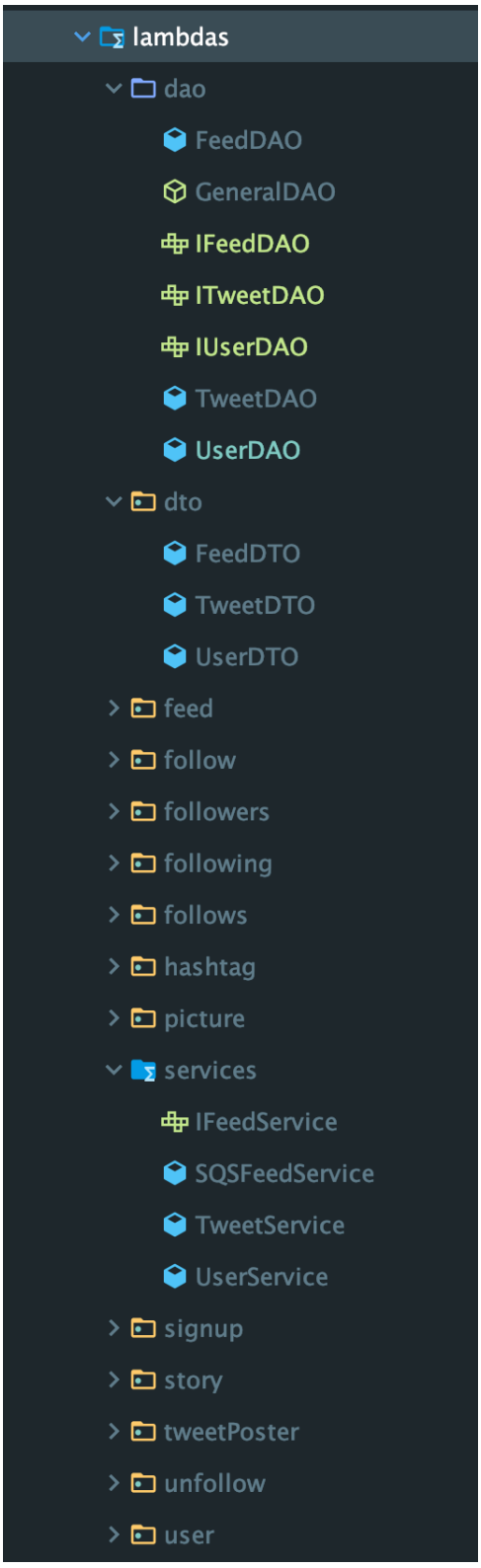
Back-end Layered Architecture

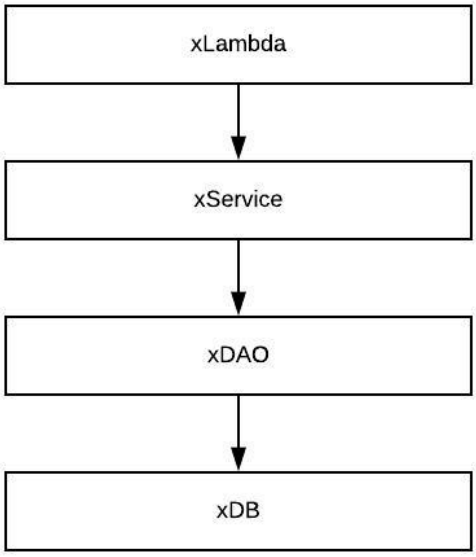
The back-end in this Twitter clone application is architected so that services are not bound or dependent on others, making it possible for changes or extensions to happen without having to change code in multiple parts of the program. For example:

- The Lambda functions are separated in packages according to each Lambda name in AWS
    - Each Lambda package contains Request objects, Response objects, and Handler objects.
- The Lambda functions reference a separate layer called Service layer, where there are two service categories: User Service and Tweet Service.
    - User Service references all user-related operations, where we need to take action to modify everything related to a user. This means registering a user in the user's table, following or unfollowing a certain user, get all followers or following, etc.
    - Tweet Service references all tweet- and feed-related operations. These operations include:
        - Tweets: posting a tweet, getting all hashtags, getting a story.
        - Feed: getting a feed based on who the user follows.
- This program also includes Data Transfer Objects (DTOs), making it possible for our Data Access Objects (DAOs) to have a model to hold data. The DTOs included in this project are:
    - UserDTO, TweetDTO, FeedDTO.
- The DAOs in this project are:
    - GeneralDAO*, UserDAO, TweetDAO, FeedDAO
        - * the GeneralDAO is another layer of abstraction that contains code for all DAOs that extend it (all of the DAOs extend GeneralDAO).

📌 This is how the project structure looks like from a package perspective.



📌 This is an example of the different back-end layers (the "x" before each entity means "any," since we can swap any database service, for example, for a new one, without a dependency problem.



- In this project, we are using DynamoDB because it is an AWS service, and all other services used in this project are AWS. This application is, however, not dependent on any such service, because of the way it was architected.