COP3503
Assignment 3

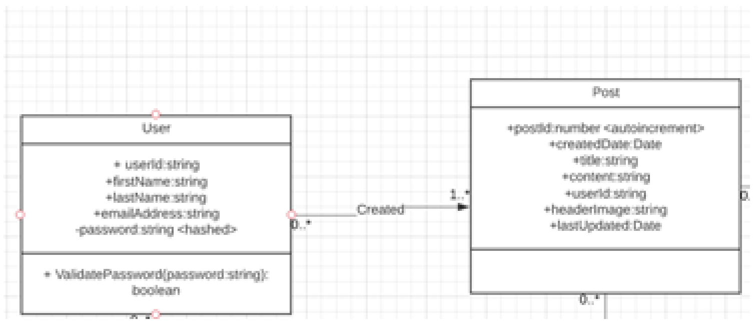
This assignment builds up on the prior one, we need to complete the back end for our application in Node / Express.

Below are the specs for an API that needs to be written in Node / Express and it must support all outlined operations. Please make sure you follow the names provided exactly for both routes and objects since the test scripts I will be using will ONLY use these names.

Furthermore since the objective of this Assignment is to support our front end development in Angular I should be able to plug in my Angular application into your Back End and get the exact same results as my own.

Write a Node application which will listen for request (web server) on port 3000 and it should provide the following routes and functionality for each route respectively.

1.) Implement the follow classes in TypeScript in your project, these classes will be used to support any and all of step #2 CRUD operations



2) Your application should support CRUD operations for all the above classes in the following EndPoints

Method	EndPoint (Route)	Functionality
[GET]	/Users	Returns a list of all users in the system, this should return all the properties of all the users except their passwords. The user's password

		should never be returned on ANY operation
[GET]	/Users/<userId>	Returns all of the given user's information except the password. (the password should be blank or not there at all when retrieving a user) Should return Status 200 (OK) under normal circumstances If a user is not found it should return an Error Message with a Status of 404 (Not Found) See below for the Error Message Syntax (JSON Object)
[POST]	/Users	Creates a new User It should check for Duplicate User ID All fields are required It should validate for valid Email Address If a duplicate ID it should return an error Message and a status of 409 (Conflict) Should return 201(Created) under normal circumstances
[PATCH]	/Users/<userId>	Should update an existing user (no updates are allowed to the userID field) If user is not found return 404 (not found) Should return 200 (OK) under normal circumstances Authentication header is required with a valid JSON Web Token Should return 401 (Not Authorized) if the request

		is not properly authenticated
[DELETE]	/Users/<userId>	Should Delete an existing user (and the user's related Posts) Should return 404 (Not Found) if the user is not found Authentication header is required with a valid JSON Web Token Should return 401 (Not Authorized) if the request is not properly authenticated and it should return a properly formatted Error Message Should return 204 (No Content) under normal circumstances)
[GET]	/Users/<userId>/<password>	Should return a valid JSON Web Token if the username and password are valid Should return 401 Un-Authorized otheriwse
[GET]	/Users/Posts/<userID>	Returns all posts for the passed in user
[GET]	/Posts	Should return all posts in the blog in sequential order of createdDate where the latest post should be first element on the Array. Should return 200 (OK) under normal circumstances
[GET]	/Posts/<postID>	Should return a specific post Should return 200 (OK) Under Normal Circumstances Should return 404 not found if the post doesn't exist
[POST]	/Posts	Creates a new post for the currently Authenticated

		User Requires Authentication header with a valid JSON Web Token Should return 401 (Not Authorized) if the request is not properly authenticated and it should return a properly formatted Error Message Should Return 201 (Created) under normal circumstances
[PATCH]	/Posts/<postID>	Updates an existing post, not changes to the postID are allowed or the creation date. Only the content and header image may be altered Requires Authentication header with a valid JSON Web Token and the currently logged in user must match the author of the post Should return 401 (Not Authorized) if the request is not properly authenticated and it should return a properly formatted Error Message Should return 404 (Not Found) if the post doesn’t exist Should return 200 (OK) under normal circumstances
[DELETE]	/Posts/<postID>	Should delete an existing post and related Comments (see below) Requires Authentication header with a valid JSON Web Token and the currently logged in user must match the author of the post Should return 401 (Not

		Authorized) if the request is not properly authenticated and it should return a properly formatted Error Message Should return 404 Not Found if the Post doesn't exist Should return 204 no content under normal circumstances.

Error Messages:
All error messages should be sent back as the following JSON Object

```
{
        "Status": 401
        "Message": "Unable to do x y z"
}
```

All objects should be sent back as JSON Objects or JSON Object Arrays

Extra Credit Implement Comments API

[GET]	/Comments/<postId>	Returns any comments associated with a Post
[POST]	/Comments/<postId>	Creates a comment associated with a given post Returns 404 if the post is not found.
[DELETE]	/Comments/<postId>/<commentId>	Deletes a comment associated with the given post / commentid

Object Schemas. Make sure the objects you send and return have this exact schema since this API needs to be interchangeable.

```
User:
 {
    "userId": "string",
    "firstName": "string",
    "lastName": "string",
    "emailAddress": "user@example.com"
    “password”: ”string”
  }
```

```
Post:
{
    "postId": 0,
    "createdDate": "2021-03-08",
    "title": "string",
    "content": "string",
    "userId": "string",
    "headerImage": "string",
    "lastUpdated": "2021-03-08"
}
```

```
Comment:
 {
    "commentId": 0,
    "comment": "string",
    "userId": "string",
    "postId": 0,
    "commentDate": "2021-03-08"
  }
```