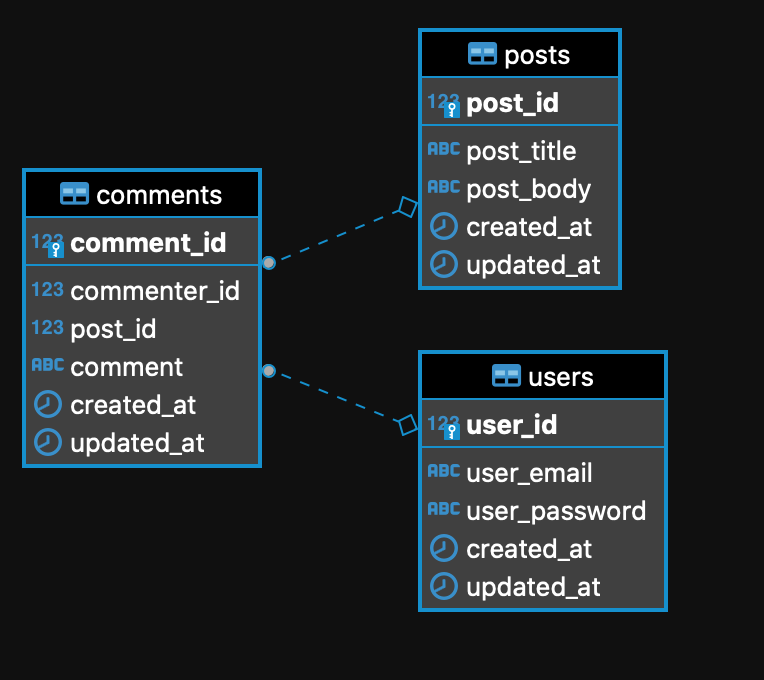
Node, Express, and PostgreSQL: Assessment

This lesson should take you between 60 and 90 minutes. If you spend longer than that on this lesson, reach out for help.

Instructions

You are a backend developer at a blogging platform called Blogful, and you've been tasked to create an API that returns data about the users, comments, and posts stored in their database. Their backend technology stack is currently Node.js/Express, PostgreSQL, and Knex. You are given the following ERD:

1:1



To complete this assessment, you will need to complete the tasks described below to get the tests to pass.

Existing files

In this lesson, all the required server routes have already been set up for you, so you won't have to edit any of the \*.router.js files. Take some time to understand the content of the existing files.

***Note:****Do not directly change any of the seed files, as the tests rely on the specific test dataset to work properly.*

You will then have to write Knex queries to complete the functions defined inside the \*.service.js and \*.controller.js files.

Tasks

1. In src/comments/comments.service.js, complete the list() function to create a Knex query that retrieves all comments from the *comments* table, selecting all columns.
   * Then, in src/comments/comments.controller.js, modify the list() function to call the service method and return a JSON of all comments in the response.
2. In src/posts/posts.service.js, complete the create() function to create a Knex query that creates a new post in the *posts* table.
   * Then, in src/posts/posts.controller.js, modify the create() route handler to call the service method to return a 201 status code along with the newly created post.
3. In src/posts/posts.service.js, complete the update() function to create a Knex query that updates a post given a body with the updated post and the postId from the URL.
   * Then, in src/posts/posts.controller.js, modify the update() route handler to call the service method and return the updated post upon success.
4. In src/posts/posts.service.js, complete the destroy() function to create a Knex query that deletes a post given a post ID.
   * Then, in src/posts/posts.controller.js, modify the delete() route handler to call the service method and return a 204 status code on successful post deletion.
5. For the /comments/commenter-count path, the API should return a count of comments from each commenter, grouped by *user\_email*, aliased to *commenter\_email*, ordered by *commenter\_email* in your result. In src/comments/comments.service.js, complete listCommenterCount() to perform a join between *comments* and *users* tables here to get the needed columns.
   * Then, in src/comments/comments.controller.js, modify the listCommenterCount() function to call the service method accordingly and return the data.
6. In src/comments/comments.service.js, complete the read() function to retrieve a comment by ID, including the following columns in your result: *comment\_id*, *comment*, *user\_email* aliased as *commenter\_email*, and *post\_body* aliased as *commented\_post*. You will need to perform a join between the *comments*, *users*, and *posts* tables here. The read() method in the controller is already completed for you.