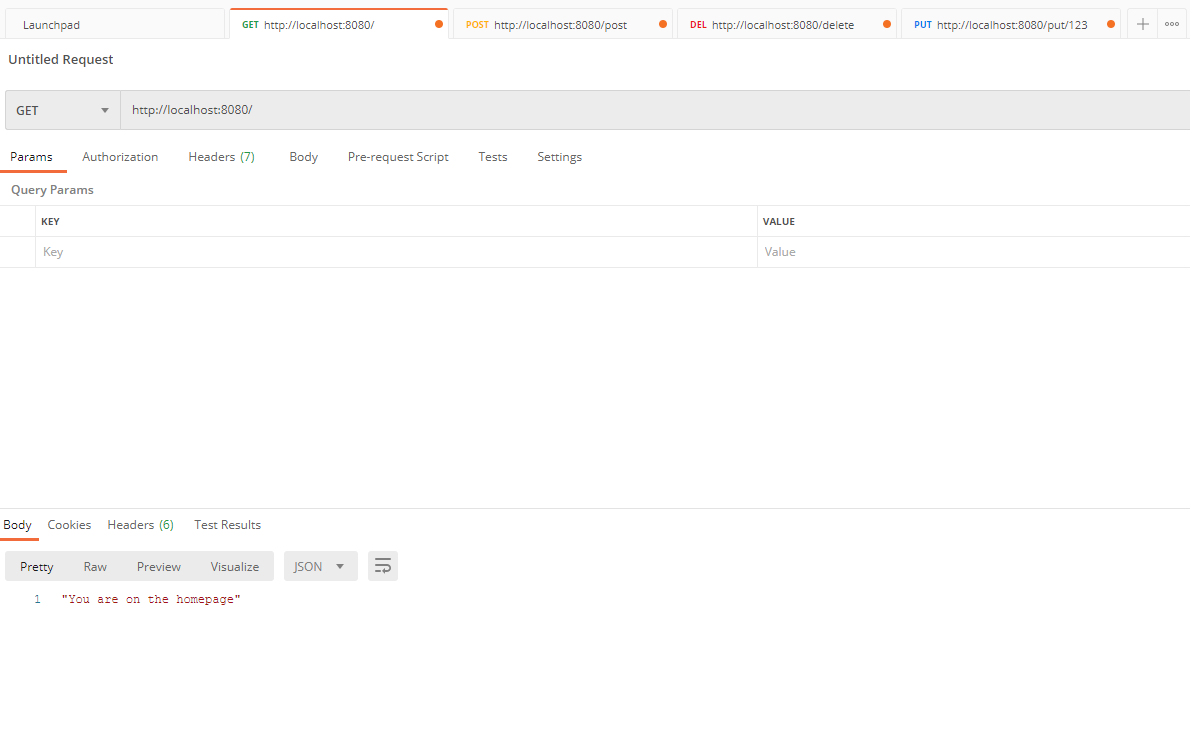
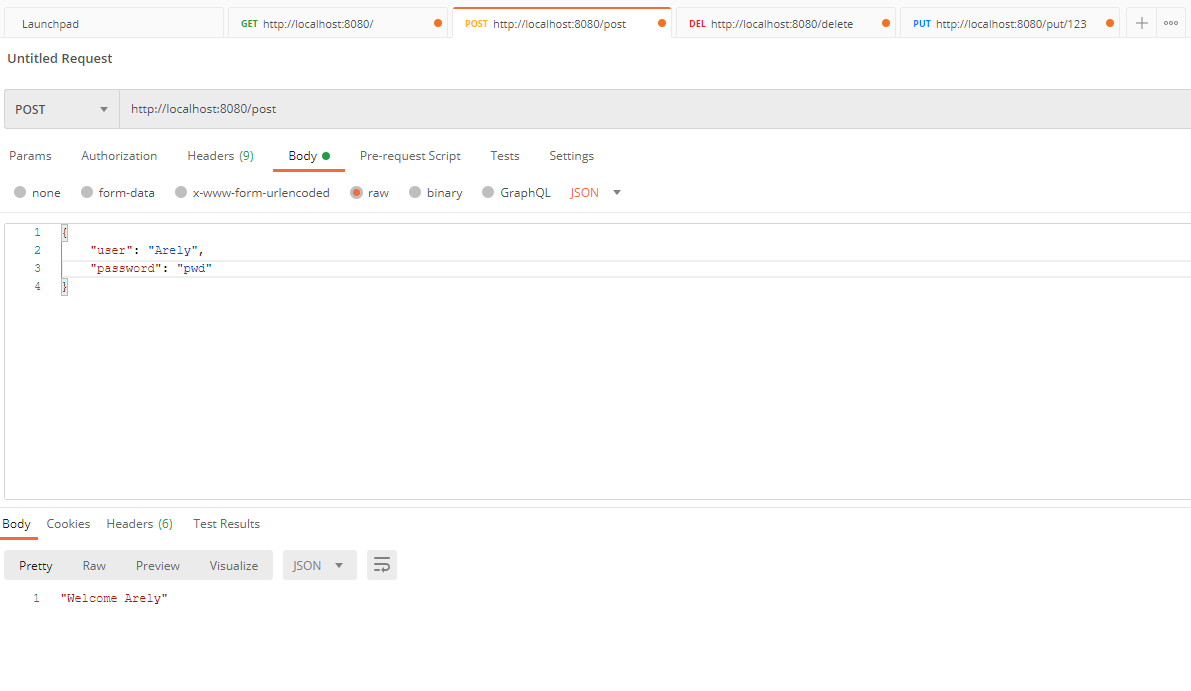
Second Partial Exam. TC2026: Web Applications Development. 2020  
  
NAME: [Arely Aceves Compean]  
GITHUB REPOSITORY: [[**https://github.com/ArelyA/Parcial-2**](https://github.com/ArelyA/Parcial-2)]

Instructions:

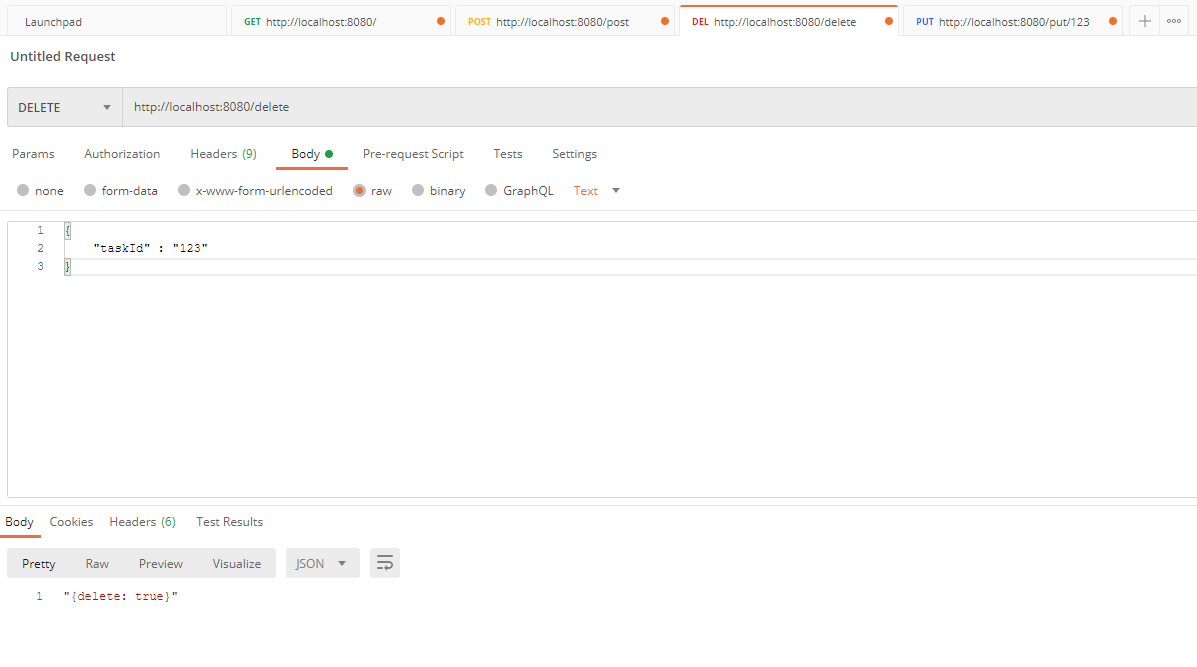
* You must have your webcam turned on.
* **Turn off your cellphone and close any social media site.**
* You are allowed to use the following during the exam:
  + Command line / Terminal / Gitbash.
  + The editor of your preference to write the coding solutions.
  + Material from class is allowed.
* Code everything from zero.
* When you finish the exam, you will need to upload to CANVAS this file.
* You cannot ask any classmate for anything. This exam is a test of how much you have learned.
* After you are done create a new github repository and upload your project from part 1. Be sure to paste your github repo in the top of this file.

**Part 1 - Express**

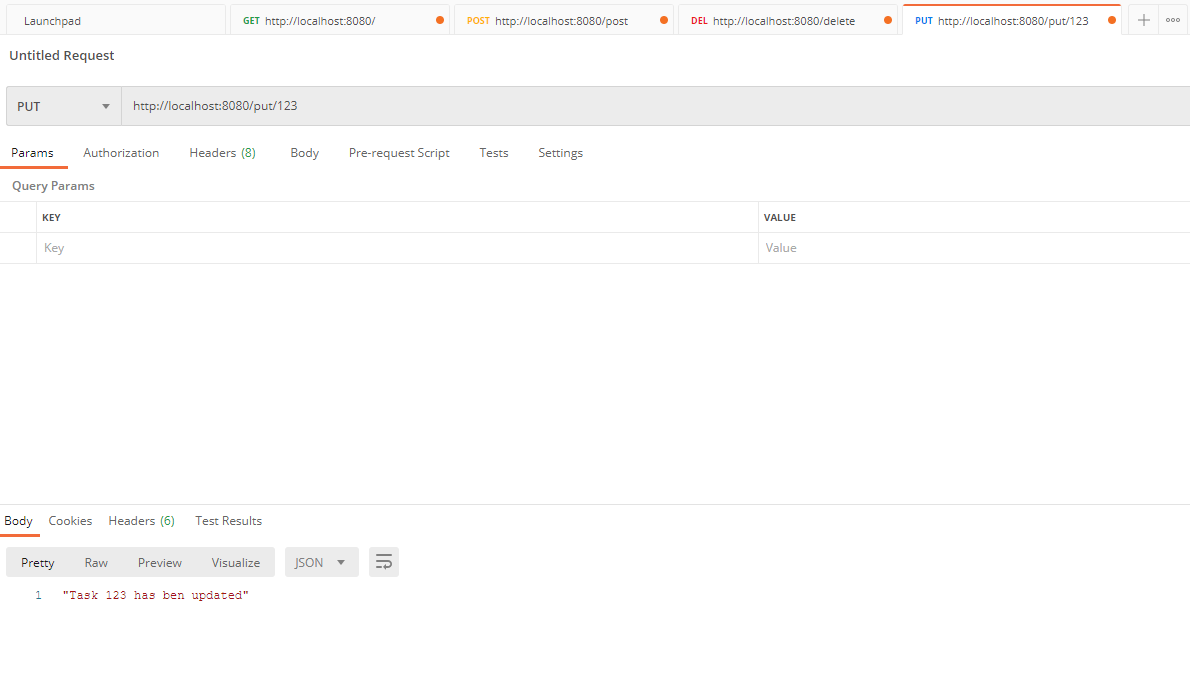
1. Create file called “server.js” and write all necessary code to have a node-express server which runs on port 5000. (Don’t forget to initialize your project).
2. Create the following routes using the necessary types of http requests. You must include a screenshot of each request with the corresponding response using any of: Postman, Rest Client, Insomnia.  
   1. ***GET*** *listening on url: “/” which replies to the client a text: “You are on the homepage”.  
        
      *
   2. ***POST*** *listening on url: “/post” which must send to the server a json object with**the following fields:* ***user, password.***

*Send a request (with random data) and reply back to the user “Welcome {user}” (where user must be the value sent in the request).* ***HINT:*** *You need to include a middleware so the server understands the json data it is receiving.  
  
*

* 1. ***DELETE*** listening on url: “/delete” which must send a json object with the following field: **taskId**

Send a request with a **taskId** and reply back to the user: “{delete: true}”  
  


* 1. ***PUT*** listening on url “/put/**{ID}**” which does not send anything in the body.

Send a request like: “/put/123 and reply back to the user: “Task 123 has been updated”  
  
**

Part 2 - HandleBars

Suppose we have the following route implemented in our server and it returns an object named **posts**

    app.get('/', function (req, res) {

        res.render('home', {

            posts: [

                {

                    author: 'Janith Kasun',

                    image: 'https://picsum.photos/500/500'

                },

                {

                    author: 'John Doe',

                    image: 'https://picsum.photos/500/500?2'

                }

            ]

        });

    });

The following is a fragment of our html which uses handlebars to show some dynamic data.

Add the corresponding handlebars scripts in order to dynamically show the image and the author. You need to add the corresponding html code for the image

<div class="posts">

    <div class="row justify-content-center">

        {{#each posts}}

        <div class="col-lg-7" style="margin-top: 50px;">

            <div class="card">

               <img src={{image}} alt={{author}}> <!-- Image should appear here -->

                <div class="card-body">

                    <h5 class="card-title">

Posted by {{author}} <!-- Author should appear here -->

</h5>

                </div>

            </div>

        </div>

        {{/each}}

    </div>

</div>

**Part 3** – **Mongo**

In this section you need to write the requested queries. You must attach a screenshot that contains the query and the result of each query. If you have issues with your local MongoDB , you can use the following web-shell:   
<https://docs.mongodb.com/manual/tutorial/getting-started/>

1. Write a query to: Create a Database called: **web-store**
2. Write a query to: Create a Collection called: **products**   
   
3. Write a query to: show the list of available collections.  
    
4. Write a query to insert a document to a Collection with the following fields/data:

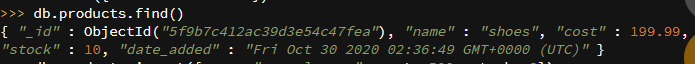
name: “shoes”

cost: 199.99

stock: 10

date\_added: [current Date]

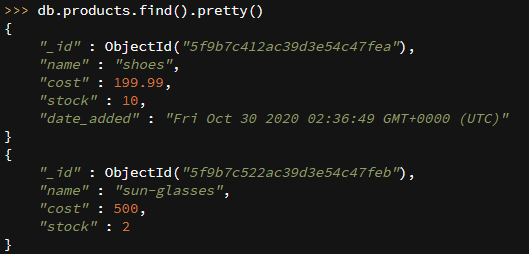
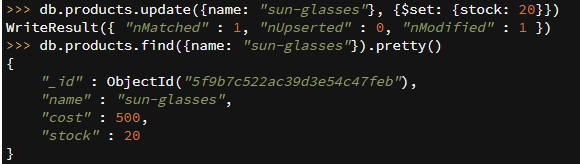
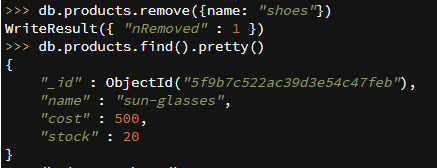


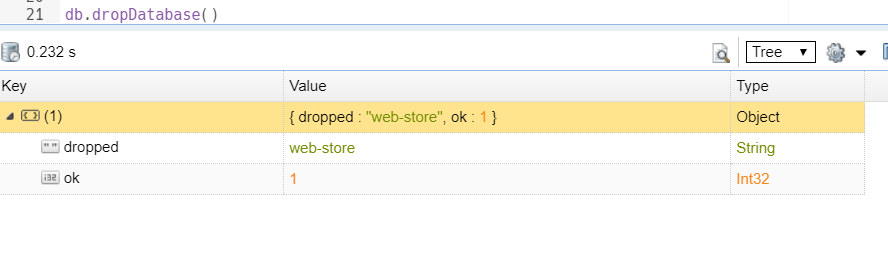
1. Write a query that: shows all available products in the **products** Collection.  
   
2. Add a second document to the **products** Collection with the following data:

name: “sun-glasses”

cost: 500

stock: 2  


1. Write a query that: shows all available products in the **products** Collection using the “pretty” mode.   
   
2. Write a query to update the “stock” of the “sun-glasses” from the current stock to 20. Also write another query to show that the update worked showing the new stock. (In total here 2 queries)  
   
3. Write a query to delete product with name “shoes” and also write a query to visualize that the product is no longer present. (In total here 2 queries)  
   
4. Write a query to drop the current database.



Bonus – 10 puntos . Menciona 1 videojuego de horror que se haya mencionado en mi podcast de #QuedateJugando. (no puede ser ni Dead Space, ni Outlast, ni Resident Evil, ni Little Nightmares, ya que esos están en la portada del podcast):

No soy mucho de jugar videojuegos y menos de terror, pero muchas tardes me la pasé viendo a un amigo jugar “The Binding of Isaac” y me hizo entender que no necesitas gráficos complicados o realistas para estar f-ed up. As a side note: no los juego, pero me gusta leer los plots y el lore de los videojuegos y uno de mis detalles favoritos es “The cake is a lie” en Portal.

“If you try and Fail, Congratulations.

Most People won’t even try”